[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 9/45 (2006.01)

[12] 发明专利说明书

专利号 ZL 200510123611.6

[45] 授权公告日 2009 年 11 月 25 日

[11] 授权公告号 CN 100562849C

[22] 申请日 2005.11.18
[21] 申请号 200510123611.6
[30] 优先权
[32] 2004.11.25 [33] JP [31] 341236/2004
[73] 专利权人 松下电器产业株式会社
地址 日本大阪
[72] 发明人 畑野文博 田中旭
[56] 参考文献
CN1521623A 2004.8.18
US6526573B1 2003.2.25
JP2004－38279A 2004.2.5
US2002/0013937A1 2002.1.31
审查员 杜 军

[74] 专利代理机构 北京律诚同业知识产权代理有限公司
代理人 徐金国 梁 挥

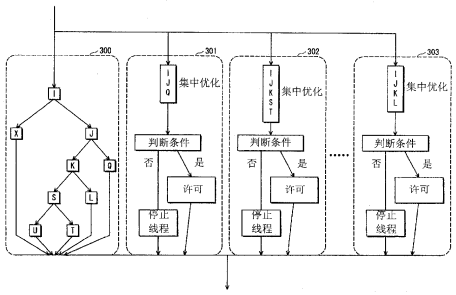
权利要求书 7 页 说明书 26 页 附图 21 页

[54] 发明名称

程序转换器件及方法、程序转换执行器件及转换执行方法

[57] 摘要

本发明公开了一种编译器器件，该器件在跟踪调度中不必采用补偿码即可产生用于可以并行执行两个或者多个指令的计算机的可执行程序。 该编译器器件产生使计算机并发执行由源程序基本直接转换得到的代码和通过优化源程序中最频繁执行路径的指令序列产生的代码的可执行程序。



1、一种程序转换器件，用于将包括条件分支的源程序转换为计算机的目标程序，所述计算机至少可以并行执行两个指令，所述程序转换器件包括：

执行路径指定单元，用于指定源程序的一个程序段中多条执行路径中的一条执行路径，所述程序段包括条件分支和所述条件分支的多个分支目标；

第一代码产生单元，用于产生对应于所述程序段中所有指令的第一代码；

第二代码产生单元，用于产生对应于指定执行路径中的指令序列的第二代码，所述第二代码包括表示如果用于选择执行路径的条件为真则继续执行序列中跟在条件分支后面的指令以及如果条件为假停止继续所述跟在条件分支后面的指令的代码，以作为对应于条件分支的代码；

第三代码产生单元，用于产生对应于所述程序段之后的源程序的随后程序段中指令的第三代码；以及

目标程序产生单元，用于产生目标程序，所述目标程序使得计算机并行执行所述第一代码和所述第二代码；并且如果所述条件为真，在第二代码后执行第三代码；而如果所述条件为假，在所述第一代码后执行所述第三代码。

2、根据权利要求 1 所述的程序转换器件，其特征在于，

所述目标程序产生单元产生还会使计算机在所述第一代码先于所述第二代码结束时停止执行所述第二代码的所述目标程序。

3、根据权利要求 1 所述的程序转换器件，其特征在于，还包括：

执行路径获取单元，用于通过所述计算机执行由所述源程序转化来的程序，而从所述计算机获取表示在所述程序段中被最频繁选择的执行路径的信息；

其中，所述执行路径指定单元指定所述被最频繁选择的执行路径。

4、根据权利要求 3 所述的程序转换器件，其特征在于，还包括

并行执行限度获取单元，用于获取表示由所述计算机可并行执行的可执行指令数量的数量 m，

其中所述执行路径获取单元还从计算机获取表示程序段中第二频繁选择的执行路径到最少选择的执行路径的信息，

所述执行路径指定单元还基于数量 m 指定第二频繁执行路径到第 n 频繁

执行路径，其中 n=m-1，

所述第二代码产生单元产生与由执行路径指定单元指定的最频繁执行路径到第 n 频繁执行路径一一对应的 n 组第二代码，并且

所述目标程序产生单元产生使计算机并行地执行第一代码和 n 组第二代码的目标代码。

5、根据权利要求 4 所述的程序转换器件，其特征在于，

所述目标程序产生单元产生还使计算机停止执行除用于相应执行路径的条件为真的一组第二代码以外的 n 组第二代码的目标程序。

6、根据权利要求 5 所述的程序转换器件，其特征在于，

所述目标程序产生单元产生使计算机不是删除而是保留任何被停止的多组第二代码的目标程序。

7、根据权利要求 1 所述的程序转换器件，其特征在于，还包括：

存储信息获取单元，用于获取表示计算机为计算机中的所有处理器内核共享一个存储器的存储器共享型还是处理器内核分别具有单独存储器的存储器分布型的存储器信息，

其中如果存储器信息显示存储器共享型，所述目标程序产生单元产生还使分别执行第一代码和第二代码的处理器内核单独处理同一变量的目标程序。

8、根据权利要求 1 所述的程序转换器件，其特征在于，还包括

用于将目标程序转换为适用于计算机的机器语言的机器语言转换单元。

9、一种程序转换和执行器件，用于将包括条件分支的源程序转化为目标程序，所述程序转换和执行器件可以并行执行至少两个指令，并包括：

执行路径指定单元，用于指定源程序的程序段中多个执行路径中的一个执行路径，所述程序段包括条件分支和所述条件分支的多个分支目标；

第一代码产生单元，用于产生对应于所述程序段中所有指令的第一代码；

执行单元，用于执行由所述源程序转化来的程序，所述程序包括第一代码；

获取单元，用于通过所述执行单元执行所述程序获取表示在程序段中被最频繁选择的执行路径的信息，其中所述执行路径指定单元指定所述被最频繁选择的执行路径；

第二代码产生单元，用于产生对应于指定的执行路径中指令序列的第二代码，所述第二代码包括表示如果用于执行路径的条件为真则继续执行序列中跟

在条件分支后的指令以及如果条件为假停止继续所述跟在条件分支后的指令的代码作为对应于条件分支的代码；

第三代码产生单元，用于产生对应于所述程序段之后的源程序的随后程序段中指令的第三代码；以及

目标程序产生单元，用于产生目标程序，所述目标程序使执行单元并行执行第一代码和第二代码；并且如果条件为真，在第二代码后执行第三代码；而如果条件为假，在第一代码后执行第三代码；

其中所述执行单元执行所述的目标程序。

10、根据权利要求 9 所述的程序转换和执行器件，其特征在于，

所述目标程序产生单元产生还会使所述执行单元在所述第一代码先于所述第二代码结束时停止执行所述第二代码的目标程序。

11、根据权利要求 10 所述的程序转换和执行器件，其特征在于，

并行执行限度获取单元，用于获取表示由程序转换和执行器件并行执行的可执行指令数量的数量 m，

其中所述执行路径获取单元还获取在程序段中第二频繁选择的执行路径到最少选择的执行路径的信息，

所述执行路径指定单元还基于数量 m 指定第二频繁执行路径到第 n 频繁执行路径，其中 n=m-1，

所述第二代码产生单元产生与由执行路径指定单元指定的最高频繁执行路径到第 n 频繁执行路径一一对应的 n 组第二代码，并且

所述目标程序产生单元产生使所述执行单元并行地执行所述第一代码和所述 n 组第二代码的目标程序。

12、根据权利要求 11 所述的程序转换和执行器件，其特征在于，

所述目标程序产生单元产生还使所述执行单元停止执行除用于相应执行路径的条件为真的一组第二代码以外的 n 组第二代码的目标程序。

13、根据权利要求 12 所述的程序转换和执行器件，其特征在于，

所述目标程序产生单元产生使所述执行单元不是删除而是保留任何被停止的多组第二代码的目标程序。

14、根据权利要求 9 所述的程序转换和执行器件，其特征在于，

如果程序转换和执行器件的存储器类型为在程序转换和执行器件中所有

处理器内核共享一个存储器的存储器共享型，则所述目标程序产生单元产生还使分别执行第一代码和第二代码的处理器内核单独处理同一变量的目标程序。

15、一种程序转换方法，用于将包括条件分支的源程序转换为用于可以并行执行至少两个指令的计算机的目标程序，所述方法包括：

执行路径指定步骤，用于指定源程序的一个程序段中多个执行路径中的一个执行路径，所述程序段包括条件分支和所述条件分支的多个分支目标；

第一代码产生步骤，用于产生对应于所述程序段所有指令的第一代码；

第二代码产生步骤，用于产生对应于指定执行路径中指令序列的第二代码，所述第二代码包括表示如果用于执行路径的条件为真则继续执行序列中跟在条件分支之后的指令以及如果条件为假停止继续所述跟在条件分支之后的指令的代码作为对应于条件分支的代码；

第三代码产生步骤，用于产生对应于所述程序段之后的源程序的随后程序段中指令的第三代码；以及

目标程序产生步骤，用于产生目标程序，所述目标程序使得所述计算机并行执行第一代码和第二代码；并且如果条件为真，在第二代码后执行第三代码；而如果条件为假，在第一代码后执行第三代码。

16、根据权利要求 15 所述的程序转换方法，其特征在于，

所述目标程序产生步骤产生还使所述计算机在所述第一代码先于所述第二代码结束时停止执行第二代码的目标程序。

17、根据权利要求 15 所述的程序转换方法，其特征在于，还包括，

执行路径获取步骤，用于通过所述计算机执行由所述源程序转化得到的程序，而从所述计算机获取表示程序段中被最频繁选择的执行路径的信息；

其中，所述执行路径指定步骤指定所述被最频繁选择的执行路径。

18、根据权利要求 17 所述的程序转换方法，其特征在于，还包括，

并行执行限度获取步骤，用于获取表示由所述计算机并行执行的可执行指令数量的数量 m，

其中所述执行路径获取步骤还从计算机获取表示在程序段中第二频繁选择的执行路径到最少选择的执行路径的信息，

所述执行路径指定步骤还基于数量 m 指定第二频繁执行路径到第 n 频繁执行路径，其中 n=m-1，

所述第二代码产生步骤产生与由执行路径指定步骤指定的最高频繁执行路径到第 n 高频繁执行路径一一对应的 n 组第二代码，并且

所述目标程序产生步骤产生使所述计算机并行地执行第一代码和所述 n 组第二代码的目标代码。

19、根据权利要求 18 所述的程序转换方法，其特征在于，

所述目标程序产生步骤产生还使所述计算机停止执行除用于相应执行路径的条件为真的一组第二代码以外的 n 组第二代码的目标程序。

20、根据权利要求 19 所述的程序转换方法，其特征在于，

所述目标程序产生步骤产生使所述计算机不是删除而是保留任何被停止的多组第二代码的目标程序。

21、根据权利要求 15 所述的程序转换方法，其特征在于，还包括：

存储信息获取步骤，用于获取表示计算机为计算机中的所有处理器内核共享一个存储器的存储器共享型还是处理器内核分别具有单独存储器的存储器分布型的存储器信息，

其中如果存储器信息显示存储器共享型，所述目标程序产生步骤产生还使分别执行第一代码和第二代码的处理器内核单独处理同一变量的目标程序。

22、根据权利要求 15 所述的程序转换方法，其特征在于，还包括：

将目标程序转换为适用于计算机的机器语言的机器语言转换步骤。

23、一种程序转换和执行方法，用于将包括条件分支的源程序转化为目标程序的程序转换和执行器件，所述程序转换和执行器件可以至少并行执行两个指令，所述方法包括：

执行路径指定步骤，用于指定源程序的一个程序段中多个执行路径中的一个执行路径，所述程序段包括条件分支和所述条件分支的多个分支目标；

第一代码产生步骤，用于在所述程序段中产生对应于所有指令的第一代码；

执行步骤，用于执行由所述源程序转化的程序，所述程序包括所述第一代码；

获取步骤，用于通过执行所述程序获取表示在程序段中被最频繁选择的执行路径的信息，其中执行路径指定步骤指定所述被最频繁选择的的执行路径；

第二代码产生步骤，用于产生对应于指定的执行路径中指令序列的第二代

码，所述第二代码包括表示如果用于执行路径的条件为真则继续执行序列中跟在条件分支之后的指令，以及如果条件为假停止继续所述跟在条件分支之后的指令的代码作为对应于条件分支的代码；

第三代码产生步骤，用于产生对应于所述程序段之后的所述源程序的随后程序段中指令的第三代码；以及

目标程序产生步骤，用于产生目标程序，所述目标程序使第一代码和第二代码并行执行；并且如果所述条件为真，在第二代码后执行第三代码；而如果所述条件为假，在第一代码后执行第三代码；

其中，所述执行步骤执行所述目标程序。

24、根据权利要求 23 所述的程序转换和执行方法，其特征在于，

所述目标代码产生步骤产生还使得在第一代码先于第二代码结束时停止第二代码执行的目标程序。

25、根据权利要求 24 所述的程序转换和执行方法，其特征在于，还包括，

并行执行限度获取步骤，用于获取表示由程序转换和执行器件并行执行的可执行指令数量的数量 m，

其中所述执行路径获取步骤还在程序段中获得表示第二频繁选择的执行路径到最少选择的执行路径的信息，

所述执行路径指定步骤还基于数量 m 指定第二频繁执行路径到第 n 频繁执行路径，其中 n=m-1，

所述第二代码产生步骤产生与由执行路径指定步骤指定的最高频繁执行路径到第 n 频繁执行路径一一对应的 n 组第二代码，并且

所述目标程序产生步骤产生使并行地执行第一代码和 n 组第二代码的目标程序。

26、根据权利要求 25 所述的程序转换和执行方法，其特征在于，

所述目标程序产生步骤产生还使停止执行除用于相应执行路径的条件为真的一组第二代码以外的 n 组第二代码的目标程序。

27、根据权利要求 26 所述的程序转换和执行方法，其特征在于，

所述目标程序产生步骤产生使得不是删除而是保留任何被停止的第二代码的目标程序。

28、根据权利要求 23 所述的程序转换和执行方法，其特征在于，

如果程序转换和执行器件的存储器类型为在程序转换和执行器件中所有处理器内核共享一个存储器的存储器共享型，则所述目标程序产生步骤产生还使分别执行第一代码和第二代码的处理器内核单独处理同一变量的目标程序。

程序转换器件及方法、程序转换执行器件及转换执行方法

技术领域

本发明涉及通过编译器进行的程序优化，并尤其涉及基于程序中执行路径的执行频率进行的优化。

背景技术

现在许多的研究工作已经开始致力于开发将源程序转换为可以在目标硬件更快运行的可执行程序的编译器上。

为了提高可执行程序的执行速度，编译器器件执行指令调度。指令调度包括重新排序程序中的指令以提高指令级并行度从而实现更快执行速度的全局调度。跟踪调度为全局调度方法的其中一种。这里，尽管在程序末端可能包括条件分支，但是在中间不包括条件分支并因此连续执行的程序中的指令序列称为基本程序块。按常规来说，对基本程序块中的指令进行重新排序以提高指令级并行度，从而减少可执行程序的执行时间。

依照跟踪调度，在其末端具有条件分支的基本程序块好像不存在条件分支一样连接到分支目标程序块之一，以产生扩展基本程序块。这样，通过对扩展基本程序块中的指令重新排序执行指令调度。

由于扩展了原始基本程序块，因此可以更灵活地执行指令调度，通过这样可以进一步减少可执行程序的执行时间。但是，在可执行程序实际执行过程中，则不可以对该扩展基本程序块的执行路径实施控制。考虑到这些，需要提供补偿代码以维持程序中数值的一致性（consistency）。在对已经进行优化的扩展基本程序块的执行路径进行控制时，该可执行程序比没有进行跟踪调度基本上直接转化源程序的可执行程序运行速度更快。这些调度技术公开于日本专利申请号 No. H11-96005 中。

上述扩展基本程序块基本上用于位于程序中频繁执行的路径的基本程序块中。

以下给出跟踪调度的具体实施例。图 20A 所示为一部分具有所示分支的

源程序的控制流程图。假设连接基本程序块 A 2001、B 2002 和 C 2003 的执行路径具有最高的执行频率。根据执行频率量向这部分源程序应用跟踪调度，例如，如图 20B 所示的结果。在扩展基本程序块 2010 中，交换基本程序块 A 2001 和 B 2002 是建立在该顺序有助于加速执行的基础上。当对该扩展基本程序块 2010 的执行路径，即基本程序块 B 2012、A 2011 和 C 2013 序列进行控制时，该总执行时间降低。

如上所述，跟踪调度对基本程序块中的指令重新排序，从而需要提供补偿代码以在对另一执行路径执行控制时的情况时保持数值的一致性。

图 20B 中的基本程序块 A’2018 用作该补偿代码。在图 20B 中，如果程序从基本程序块 B 2012 直接分支到图 20A 所示的基本程序块 D2004，则基本程序块 A 2001 的操作将终止并错过。这样，为了保持图 20A 中连接基本程序块 A2001、B2002、D2004 和 E2005 的执行路径的数值一致性，将基本程序块 A’2018 插入作为对应于基本程序块 A2001 的补偿代码。

如果程序包括更复杂的条件分支，则补偿代码也变得更加复杂。在某些情况下，在对包括补偿代码的程序路径进行控制时，程序的运行可能比预想的慢。因此，提供补偿代码可能会使总执行时间增加。

发明内容

为了解决上述问题，本发明目的在于提供用于通过在具体执行路径形成扩展基本程序块并且不必通过采用补偿代码优化该扩展基本程序块产生程序的程序转换器件。

通过用于将包括条件分支的源程序转换为用于至少可以并行执行两个指令的计算机的目标程序的程序转换器件实现上述目的，该程序转换器件包括：用于指定源程序的一个程序段中多个执行路径中的一个执行路径的执行路径指定单元，所述程序段包括条件分支和所述条件分支的多个分支目标；用于产生对应于所述程序段中所有指令的第一代码的第一代码产生单元；用于产生对应于指定执行路径中指令序列的第二代码的第二代码产生单元所述第二代码包括表示如果用于执行路径的条件为真则继续执行序列中跟在条件分支之后的指令以及如果条件为假停止继续所述指令的代码作为对应于条件分支的代码；用于产生对应于源程序随后部分的指令的第三代码的第三代码产生单元；

以及用于产生使得计算机并行执行第一代码和第二代码并且如果条件为真在第二代码后执行第三代码而如果条件为假在第一代码后执行第三代码的目标程序的目标程序产生单元。

用在这里的术语“对应于”表示代码基本上具有和源程序中指令一样的内容。但是应该注意要访问的寄存器依赖于计算机存储器的类型而改变。此外，执行路径表示连续执行的指令序列。当程序在条件分支处分支时，执行路径表示那个条件分支的多个分支目标中的单独的一个。由目标程序产生单元产生的目标程序为中间代码或者准备在计算机上运行的可执行程序。该中间代码表示在将源程序转换为目标程序的过程中产生从而便于通过程序转换器件进行代码处理的代码，并且该代码具有对应于源程序的内容。

根据上述的架构，目标程序使计算机中的一个处理器内核执行由源程序不经过优化基本上直接转化得到的代码，并且该计算机中的另一处理器内核用于执行通过优化指定执行路径中指令序列产生的第二代码。

这样，在不必采用对控制选择另一执行路径时保持数值一致性通常需要的补偿代码就可以产生关于指定执行路径已经进行优化的程序。而且，在控制选择指定执行路径时，第二代码运行速度比第一代码快，这加速了第三代码的开始。因此，减少了总执行时间。而且，由于第一处理器内核执行对应于原始源程序的第一代码因此可以保持数值的一致性。

这里，目标程序产生单元可以还使计算机在第一代码先于第二代码结束时停止执行第二代码的目标程序。

根据所述架构，组织目标程序使得在第一代码先于第二代码结束时执行第二代码的处理器内核停止执行，然后向该处理器内核分配另一线程。这有助于有效资源利用。

这里，所述程序转换器件还包括用于通过计算机执行由源程序基本上直接转化得到的程序从计算机获取表示在所述程序段中被最频繁选择的执行路径的信息的执行路径获取单元，其中执行路径指定单元指定最频繁的执行路径。

根据该结构，优化所述最频繁执行路径中的指令序列。因此，当控制选择所述执行路径时，可以减少程序的执行时间。

这里，程序转换器件还包括用于获取表示由计算机并行执行的可执行指令数量的数量 m 的并行执行限度获取单元，其中所述执行路径获取单元还从计

算机获取表示在程序段中第二频繁执行路径到最少频繁执行路径的信息，所述执行路径指定单元还基于数量 m 指定第二频繁执行路径到第 n 频繁执行路径，其中 n=m-1，所述第二代码产生单元产生与由执行路径指定单元指定的最高频繁执行路径到第 n 频繁执行路径一一对应的 n 组第二代码，并且所述目标程序产生单元产生使计算机单独并行地执行第一代码和 n 组第二代码的目标代码。

根据该结构，两个或者多个具有高执行频率的执行路径可以作为单独线程执行，这样可以减少总执行时间。

这里，所述目标程序产生单元产生还使计算机停止执行除用于相应执行路径的条件为真的一组第二代码以外的 n 组第二代码的目标程序。

根据该结构，，目标程序组织在控制选择一条执行路径时使得执行那条执行路径线程的处理器内核停止其他线程。

这里，目标程序产生单元产生使计算机不是删除而是保留任何被停止的多组第二代码的目标程序。

根据该结构，当下一线程和当前线程一样而仅仅是操作数据不同时，由于保留了当前线程因此仅需要将操作数据传递给所述处理器内核。这省去了每次向处理器内核传递线程和处理数据的麻烦，通过这样可以减少该程序的执行时间。

这里，程序转换器件还包括用于获取表示计算机为计算机中的所有处理器内核共享一个存储器的存储器共享型还是处理器内核分别具有单独存储器的存储器分布型的存储器信息的存储信息获取单元，其中如果存储器信息显示存储器共享型，所述目标程序产生单元产生还使分别执行第一代码和第二代码的处理器内核单独处理同一变量的目标程序。

为了单独处理同一变量装置（a same variable means），当第一代码和第二代码在源程序中参考同一变量时，执行第一代码的处理器内核和执行第二代码的处理器内核在不同的寄存器存储所述变量。

根据该结构，在存储器共享型计算机中可以确保根据程序执行的操作结果。

这里，所述程序转换器件还包括用于将目标程序转换为适用于计算机的机器语言的机器语言转换单元。

根据该结构，如果目标程序为中间代码，所述中间代码还可以转换为以适

用于计算机的机器语言书写的可执行程序。

还可以通过用于将包括条件分支的源程序转化为目标程序的程序转换和执行器件实现所述的目的，所述程序转换和执行器件可以并行执行至少两个指令，所述器件包括：用于指定在源程序的一个程序段中多个执行路径中的一个执行路径的执行路径指定单元，所述程序段包括条件分支和所述条件分支的多个分支目标；用于产生对应于所述程序段中所有指令的第一代码的第一代码产生单元；用于执行基本上为源程序直接转化的程序的执行单元，所述程序包括第一代码；用于通过所述执行单元执行所述程序获取表示在程序段中被最频繁选择的执行路径的信息的获取单元，其中执行路径指定单元指定所述最频繁的执行路径；用于产生对应于指定的执行路径中指令序列的第二代码的第二代码产生单元所述第二代码包括表示如果用于执行路径的条件为真则继续执行序列中跟在条件分支后面的指令以及如果条件为假停止继续所述指令的代码作为对应于条件分支的代码；用于产生对应于源程序随后程序段中指令的第三代码的第三代码产生单元；以及用于产生使执行单元并行执行第一代码和第二代码并且如果条件为真在第二代码后执行第三代码而如果条件为假在第一代码后执行第三代码的目标程序的目标程序产生单元，其中所述执行单元执行目标程序。

根据所述结构，可以产生程序并且执行程序的程序转换和执行器件可以产生在控制选择频繁执行路径时运行更快的程序。

如上所述，越复杂的控制流程图就要求越复杂的补偿代码。在采用即时编译（just-in-time）即，动态转换的编译器件中，为了提高在连续分析和执行每行代码的解释程序中部分代码的执行性能，所述补偿代码的产生会浪费时间。根据本发明，由于不需要产生补偿代码因此不会产生所述问题。

这里，目标程序产生单元可以产生还使执行单元在第一代码先于第二代码结束时停止执行第二代码的目标程序。

根据所述结构，目标程序组织为在第一代码先于第二代码结束时使执行第二代码的处理器内核停止执行，然后向该处理器内核分配另一线程。这有助于有效资源利用。

这里，程序转换和执行器件还包括用于获取表示由程序转换和执行器件并行执行的可执行指令数量的数量 m 的并行执行限度获取单元，其中所述执行

路径获取单元还获取表示在程序段中第二频繁执行路径到最少频繁执行路径的信息，所述执行路径指定单元还基于数量 m 指定第二频繁执行路径到第 n 频繁执行路径，其中 n=m-1，所述第二代码产生单元产生与由执行路径指定单元指定的最高频繁执行路径到第 n 频繁执行路径一一对应的 n 组第二代码，并且所述目标程序产生单元产生使执行单元单独并行地执行第一代码和 n 组第二代码的目标程序。

根据所述结构，两个或者多个具有高执行频率的执行路径可以作为单独线程执行，这样可以减少总执行时间。

这里，所述目标程序产生单元产生还使执行单元停止执行除用于相应执行路径的条件为真的一组第二代码以外的 n 组第二代码的目标程序。

根据所述结构，所述目标程序组织为在执行一个线程的条件为真时使其他处理器内核停止执行其他线程，然后向那些处理器内核分配下面的线程。这有助于资源的有效利用。

这里，目标程序产生单元产生使执行单元不是删除而是保留任何被停止的多组第二代码的目标程序。

根据所述结构，当下一线程和当前线程一样而仅仅是操作数据不同时，由于保留了当前线程因此仅需要将操作数据传递给所述处理器内核。这省了每次向处理器内核传递线程和处理数据的麻烦，通过这样可以减少该程序的执行时间。

这里，如果程序转换和执行器件的存储器类型为在程序转换和执行器件中所有处理器内核共享一个存储器的存储器共享型，则所述目标程序产生单元产生组还使分别执行第一代码和第二代码的处理器内核单独处理同一变量的目标程序。

根据所述结构，所述目标程序组织为根据程序转换和执行器件为存储器共享型还是存储器分布型适当地向寄存器赋值。

还可以通过用于将包括条件分支的源程序转换为可以并行执行至少两个指令的计算机的目标程序的程序转换方法实现所述的目的，所述方法包括：指定一源程序中一个程序段的多个执行路径中的一个执行路径的执行路径指定步骤，所述程序段包括条件分支和所述条件分支的多个分支目标；产生对应于所述程序段所有指令的第一代码的第一代码产生步骤；产生对应于指定执行路

径中指令序列的第二代码的第二代码产生步骤，所述第二代码包括表示如果用于执行路径的条件为真则继续执行序列中跟在条件分支之后的指令以及如果条件为假停止继续该指令的代码作为对应于条件分支的代码；产生对应于在源程序随后程序段中指令的第三代码的第三代码产生步骤；以及产生使得计算机并行执行第一代码和第二代码并且如果条件为真在第二代码后执行第三代码而如果条件为假在第一代码后执行第三代码的目标程序的目标程序产生步骤。

根据所述方法，可以产生用于并行执行第一代码和通过优化指定执行路径产生的第二代码的目标程序。

这里，所述目标程序产生步骤产生还使计算机在所述第一代码先于所述第二代码结束时停止执行第二代码的目标程序。

根据所述方法，，目标程序组织为使在第一代码先于第二代码结束时执行第二代码的处理器内核停止执行。

这里，所述程序转换方法还包括通过执行由源程序基本上直接转化得到的程序从计算机获取表示程序段中使用频率最高的执行路径的信息的执行路径获取步骤，其中所述执行路径指定步骤指定最高频率的执行路径。

根据所述方法，所述目标程序组织为并行执行第一代码和通过优化最频繁执行路径中指令获得的第二代码。

这里，程序转换方法还包括获取表示由计算机并行执行的可执行指令数量的数量 m 的并行执行限度获取步骤，其中所述执行路径获取步骤还从计算机获取表示在程序段中被第二频繁选择执行路径到最少频繁选择执行路径的信息，所述执行路径指定步骤还基于数量 m 指定第二频繁执行路径到第 n 频繁执行路径，其中 n=m-1，所述第二代码产生步骤产生与由执行路径指定步骤指定的最频繁执行路径到第 n 频繁执行路径一一对应的 n 组第二代码，并且所述目标程序产生步骤产生使计算机单独并行地执行第一代码和所述 n 组第二代码的目标代码。

根据所述方法，所述目标程序组织为并行执行第一代码和通过优化多个频繁执行路径产生的多组第二代码。

这里，所述目标程序产生步骤产生进还使计算机停止执行除用于相应执行路径的条件为真的一组第二代码以外的 n 组第二代码的执行的目标程序。

根据所述方法，，目标程序组织使在当控制选择一条执行路径时，执行所

述执行路径的处理器内核停止其他线程。

这里，目标程序产生步骤产生使计算机不是删除而是保留任何被停止的多组第二代码的目标程序。

根据所述方法，可以产生能够保留线程用于进一步应用的目标程序。

这里，程序转换方法还包括获取表示计算机为计算机中的所有处理器内核共享一个存储器的存储器共享型还是处理器内核分别具有单独存储器的存储器分布型的存储器信息的存储信息获取步骤，其中如果存储器信息显示存储器共享型，所述目标程序产生步骤产生还使分别执行第一代码和第二代码的处理器内核单独处理同一变量的目标程序。

根据所述方法，在存储器共享型计算机中可以保证根据程序执行操作的结果。

这里，所述程序转换方法还包括将目标程序转换为适用于计算机的机器语言的机器语言转换步骤。

根据所述方法，如果所述目标程序为中间代码，所述中间代码还可以转换为用适用于计算机的机器语言书写的可执行程序。

还可以通过用在用于将包括条件分支的源程序转化为目标程序的程序转换和执行器件的程序转换和执行方法实现所述的目的，所述程序转换和执行器件可以至少并行执行两个指令，所述方法包括：指定在源程序的一个程序段中多个执行路径中的一个执行路径的执行路径指定步骤，所述程序段包括条件分支和所述条件分支的多个分支目标；产生对应于所述程序段中所有指令的第一代码的第一代码产生步骤；执行为源程序基本上直接转化的程序的执行步骤；通过执行所述程序获取表示在程序段中被最频繁选择的执行路径的信息的获取步骤，其中执行路径指定步骤指定所述最频繁执行路径；产生对应于指定的执行路径中指令序列的第二代码的第二代码产生步骤，所述第二代码包括表示如果用于执行路径的条件为真则继续执行序列中随在条件分支之后的指令以及如果条件为假停止继续所述指令的代码作为对应于条件分支的代码；产生对应于源程序随后程序段中指令的第三代码的第三代码产生步骤；以及产生使第一代码和第二代码并行执行并且如果所述条件为真在第二代码后执行第三代码而如果所述条件为假在第一代码后执行第三代码的目标程序的目标程序产生步骤，其中所述执行步骤执行目标程序。

根据所述方法，在运行期间可以产生用于并行执行第一代码和通过优化最高频率执行路径得到的第二代码的目标程序。

这里，目标代码产生步骤产生还使在第一代码先于第二代码结束时停止第二代码执行的目标程序。

根据所述方法，目标程序组织在第一代码先于第二代码结束时使执行第二代码的处理器内核停止执行。

这里，程序转换和执行方法还包括获取表示由程序转换和执行器件并行执行的可执行指令数量的数量 m 的并行执行限度获取步骤，其中所述执行路径获取步骤还获取表示在程序段中被第二频繁选择的执行路径到被最少频繁选择的执行路径的信息，所述执行路径指定步骤还基于数量 m 指定第二频繁执行路径到第 n 频繁执行路径，其中 n=m-1，所述第二代码产生步骤产生与由执行路径指定步骤指定的最高频繁执行路径到第 n 频繁执行路径一一对应的 n 组第二代码，并且所述目标程序产生步骤产生使并行单独地执行第一代码和 n 组第二代码的目标程序。

根据所述方法，目标程序组织为以单独线程执行两个或者以上频繁执行路径。

这里，所述目标程序产生步骤产生还使停止执行除用于相应执行路径的条件为真的一组第二代码以外的 n 组第二代码的目标程序。

根据所述的方法，，所述目标程序组织在用于执行一个线程的条件为真时使其他处理器内核停止执行其他线程。

这里，所述目标程序产生步骤产生使不是删除而是保留任何被停止的第二代码的目标程序。

根据所述方法，可以产生能够保留线程用于将来使用的目标程序。

这里，如果程序转换和执行器件的存储器类型为在程序转换和执行器件中所有处理器内核共享一个存储器的存储器共享型，则所述目标程序产生步骤产生还使分别执行第一代码和第二代码的处理器内核单独处理同一变量的目标程序。

根据所述方法，根据存储器类型为共享型还是分布型来产生目标程序。

附图说明

通过结合说明本发明具体实施方式的附图的以下说明，将使本发明的这些和其它目的、优点和特征变得更加明显。

附图中：

图 1 所示为根据本发明的实施方式的编译器件结构框图；

图 2 所示为用于解释本发明概念的控制流程图；

图 3 所示为表示本发明概念的图示；

图 4A 和图 4B 所示为处理器内核和存储器之间的关系图；

图 5A 和图 5B 所示为用于该实施方式的源程序及其控制流程图；

图 6 所示为基本直接将图 5A 所示的源程序转换为汇编码的代码；

图 7 所示为在目标硬件为存储器共享型的情况，对应于执行路径 500->501->502 的代码；

图 8 所示为在目标硬件为存储器共享型的情况，对应于执行路径 500->501->503 的代码；

图 9 所示为在目标硬件为存储器共享型的情况，对应于执行路径 500->504 的代码；

图 10 所示为在目标硬件为存储器共享型的情况的线程控制代码；

图 11 所示为在目标硬件中可以并行执行处理器内核的数量未知的情况的线程控制代码；

图 12 所示为在目标硬件为存储器分布型的情况，对应于执行路径 500->501->502 的代码；

图 13 所示为在目标硬件为存储器分布型的情况，对应于执行路径 500->501->503 的代码；

图 14 所示为在目标硬件为存储器分布型的情况，对应于执行路径 500->504 的代码；

图 15 所示为检测执行频率操作的流程图；

图 16 所示为关于目标硬件的硬件配置的判断操作的流程图；

图 17 所示为目标硬件为存储器分布型的情况下可执行程序步骤流程图；

图 18 所示为根据本发明的实施方式的程序转换和执行器件的框图；

图 19 所示为产生可执行程序的操作的流程图；

图 20A 和图 20B 所示为用于说明现有技术中跟踪调度的控制流程图；以

及

图 21 所示为目标硬件为存储器分布型的情况下的线程控制代码。

具体实施方式

以下通过参照附图说明根据本发明作为程序转换器件或者程序转换和执行器件的编译器件的实施方式。

第一实施方式

本发明第一实施方式的编译器器件产生用于存储器共享型计算机的可执行程序。

（概述）

首先，以下通过参考图 2 和 3 给出本发明的概述。

假设该编译器器件将一部分具有如图 2 所示控制流图的分支的源程序转换为可执行程序。

在图中，程序块 I 200、J 202、K 203、L 206、Q204、S 205、T 208 和 X 201 均为基本程序块。如上所述，基本程序块为尽管末端包括分支但中间不包括分支的指令序列。通过编译器器件产生的可执行程序设计为用于可以并行执行两个或者更多指令的计算机中。

图 2 中的控制流图包括五个执行路径，即，执行路径 I 200->J 202->Q 204、执行路径 I 200->J 202->K 203->S 205->T 208、执行路径 I 200->X 201、执行路径 I 200->J 202->K 203->S 205->U 207，以及执行路径 I 200->J 202->K 203->L 206。这些执行路径具有以该顺序降低的执行频率。

这样，以可执行程序的形式产生对应于这些执行路径中的一个或者多个频繁执行路径的指令序列的代码。此外，以可执行形式产生直接对应于原始源程序的代码。然后产生促使处理器内核（processor element）并行地执行对应于频繁执行路径的代码和对应于源程序的代码的可执行程序。图 3 详细示出可执行程序的步骤。如图所示，该可执行程序使得第一处理器内核执行由源程序基本上直接转换来的可执行形式的线程 300，第二处理器内核执行对应于最频繁执行路径的线程 301，第三处理器内核执行对应于第二高频繁执行路径的线程 302 等等。因此，只要可以并行执行的处理器内核数量和可创建的线程的数量允许，该可执行程序组织为使得多个处理器内核启动并并行地执行线程。在用

于执行一个线程的条件为真时，该可执行程序还使得执行该线程的处理器内核停止其他线程并执行提交(commitment)以反映该线程的操作结果。

这使得程序中不必使用补偿代码。并发执行的线程包括基本直接将源程序转换为可执行形式的线程 300，可以保持程序中的数值一致性。而且，在对对应于线程 301 到 303 的执行路径其中之一进行控制时，可以比仅执行线程 300 时获得更快的执行结果。因此可以减少了总执行之间。

（结构）

图 1 所示为在第一实施方式中编译器器件 100 的结构框图。如图所示，该编译器器件 100 大致由分析单元 101、执行路径指定单元 102、优化单元 103、以及代码转换单元 104 构成。

实际上可以通过包括 MPU（微处理单元）、ROM（只读存储器）、RAM（随机存储器）和硬盘器件构成的计算机系统实现该编译器 100。该编译器器件 100 根据存储在硬盘器件或者 ROM 中的计算机程序产生预期可执行程序（intended executable program）。采用 RAM 执行单元之间的数据传输。

该分析单元 101 分析源程序 110 中的分支和执行内容，并获取写在源程序 110 中诸如“分支”和“重复”的信息。该分析单元 101 向执行路径指定单元 102 输出作为分析结果获得的分析信息 105。

执行路径指定单元 102 从分析单元 101 接收包括源程序 110 中的执行路径识别码的分析信息 105。该执行路径指定单元 102 在以可执行形式转换的源程序 110 中获取关于执行路径的执行频率的执行频率信息 140。基于这些信息，执行路径指定单元 102 指定多个执行路径中的一个或者多个频繁执行路径，并通知指定的执行路径的优化单元 103。

优化单元 103 对于可执行程序的产生执行优化，诸如优化源程序 110 中指令的顺序。详细地，基于从分析单元 101 和执行路径指定单元 102 接收的信息，该优化单元 103 优化每个指定执行路径的指令顺序从而不会产生任何向其他执行路径的分支。

代码转换单元 104 以通过将优化单元 103 优化后的代码赋予目标硬件 130 的单独处理器内核的形式产生应用于目标硬件 130 的可执行程序 120。代码转换单元 104 向目标硬件 130 输出该可执行程序 120。

然后在目标硬件 130 执行该可执行程序 120。将关于作为执行结果产生的

执行路径的信息发送给执行路径指定单元 102 作为执行频率信息 140。这里，执行频率信息 140 表示已经在执行中采用了由分支形成的哪个执行路径。如果可执行程序 120 包括循环，那么该执行频率信息还表示在执行中每个单独执行路径被采用了多少次。

目标硬件 130 具有多个处理器内核，因此可以并行执行两个或者多个指令。目标硬件 130 的存储器类型或者为存储器共享型（memory sharing）或者为存储器分布型（memory distribution）。在第一实施方式中，假设目标硬件 130 为存储器共享型。

以下简单说明存储器共享型和存储器分布型。

如图 4A 所示，在存储器共享型中，多个处理器内核 400 到 402 连接到单个存储器 403 上。每个处理器内核 400 到 402 将来自存储器 403 的必要数据读取到其本身的寄存器中，采用该寄存器中的数据执行操作，并根据操作的结果更新存储在存储器 403 中的数据。

另一方面，如图 4B 所示，在存储器分布型中，多个处理器内核 410 到 412 分别连接到存储器 413 到 415 上。通过每个处理器内核 410 到 412 执行的程序设定为向所有存储器 413 到 415 反映处理器内核的操作结果。例如，在处理器内核 410 产生操作结果时，采用该操作结果不但更新存储在存储器 413 中的数据而且更新存储在存储器 414 和 415 中的数据。

尽管在上述两个实施例中处理器内核的数量为 3，但是处理器内核的数量不限于此。

（数据）

输入到编译器器件 100 的数据包括源程序 110、执行频率信息 140 和关于目标硬件 130 的硬件配置的信息。以下给出这些数据的说明。

执行频率信息 140 由通过分析单元 101 指定的执行路径识别码和表示通过识别码识别的执行路径在目标硬件 130 或者可以执行可执行程序的其他硬件上实际执行了多少次的信息。将获得最大执行次数的执行路径设定为具有最高执行频率的执行路径，将获得第二大次数的执行路径设定为具有第二高执行频率的执行路径等等。执行频率信息 140 存储在目标硬件 130 的 RAM 上，并将该信息发送给编译器器件 100 而且存储在其 RAM 内。

关于目标硬件 130 的硬件配置信息包括存储器信息和并行执行信息。存储器信息表示目标硬件 130 的存储器类型。如果目标硬件 130 为存储器共享型则将存储器信息设定为 0，并且如果目标硬件 130 为存储器分布型则设定为 1。将该存储器信息从目标硬件 130 发送给编译器器件 100 并存储在该编译器器件 100 的 RAM 中。并行执行信息表示通过目标硬件 130 能够并行执行的指令的数量，即，目标硬件 130 中处理器内核的数量。将该并行执行信息从目标硬件 130 发送给编译器器件 100 并还存储在编译器 100 的 RAM 中。

作为一个实施例，图 5A 所示为记录的源程序 110。

在第一实施方式中，作为源程序 110 的实施例，通过编译器器件 100 转换图 5A 所示的源程序段 510。以下说明源程序段 510 的内容以及通过编译器器件 100 由源程序段 510 产生的代码。

首先说明如图 5A 所示的源程序段 510 的内容。注意为了执行该源程序段 510 的内容的至少一部分通过编译器器件 100 产生图 6 到 10 所示的代码。

源程序段 510 为在源程序 110 中重复很多次的源程序 110 的一部分。图 5B 示出该源程序段 510 的控制流程图。通过参照该控制流程图说明源程序 510 的内容。

首先，指令块 500 对 a 和 b 加和并在 x 中存储产生的和。分支块 505 判断是否 $x \geqq 0$，如果 $x<0$（505：是），控制进入指令块 504，在该指令块 504 中将负 x 存储在 y 中。如果 $x \geqq 0$（505：否），控制进入指令块 501，该指令块块 501 中从 x 中减去 c 并将差存储在 y 中。

此后，分支块 506 判断是否 $x \geqq 10$。如果 $x \geqq 10$（506：是），控制进入指令块 502，该指令块从 y 中减去 10 并将差存储在 y 中。如果 $x<10$（506：否），控制进入指令块 503，该指令块 503 中对 x 加 10 并且将相加后的和存储在 y 中。

这里，在源程序段 510 的前一部分对 a、b 和 c 已经进行赋值指定。假设在源程序段 510 中由条件分支产生的三个执行路径中，执行路径 551 具有最高的执行频率并且执行路径 552 具有第二高的执行频率。该执行频率的信息可以通过在目标硬件 130 中执行没有优化基本上由源程序 110 直接转换来的可执行程序获得。

图 6 到图 10 所示的代码为表示从编译器器件 100 输出程序的汇编码，并

且该汇编码基于图 5A 所示的源程序段 510 产生。图 10 所示的线程 1000 为主线程。在图 7、8 和 9 中分别示出的线程 700、800 和 900 用于该主线程中。尽管没有在代码中示出，但是这些线程还是构造为通过目标硬件 130 中单独的处理器内核执行。

图 6 所示的线程 600 为表示没有优化的源程序段 510 的汇编码。尽管在图 10 中未示出，但是线程 600 还是包括在作为主线程的线程 1000 中。

这里假设每个线程中的代码行从第一行起顺序执行。在下文中进行说明对应于每行代码的指令的含义。

在线程 600 中，代码 601、609、617、622、627 和 632 为用于在程序中表示分支目标的标号代码。

代码 602 到 608 对应于图 5B 中的程序块 500 和 505。

代码 610 到 616 对应于图 5B 中的程序块 501 和 506。

代码 618 到 621 对应于图 5B 中的程序块 502。

代码 623 到 626 对应于图 5B 中的程序块 503。

代码 628 到 631 对应于图 5B 中的程序块 504。

代码 633 到 634 相当于线程 600 的结束操作。

另一方面，图 7 到图 9 分别示出的线程 700、800 和 900 均对应于频繁执行路径中的指令序列。

图 7 所示为通过优化在具有最高执行频率的执行路径 551 中指令序列产生的线程 700。

在线程 700 中，代码 701、713 和 716 为标号代码。

代码 702 到 712 对应于没有任何指向其他执行路径的分支的程序块 500、501 和 502，并包括对应于程序块 505 和 506 的代码的代码，该代码表示控制是否选择执行路径 551 的二元判定。

当控制选择执行路径 551 时，代码 714 和 715 停止其他线程 800 和 900。

代码 717 和 718 相当于线程 700 的结束操作。

图 8 所示为通过优化具有第二高执行频率的执行路径 552 中的指令序列产生的线程 800。

在线程 800 中，代码 801、814 和 817 为标号代码。

代码 802 到 813 对应于没有任何指向其他执行路径的分支的程序块 500、

501 和 503。

在控制选择执行路径 552 时，代码 815 和 816 停止其他线程 700 和 900。

代码 818 和 819 相当于线程 800 的结束操作。

图 9 示出通过优化连接程序块 500 和 504 的执行路径中的指令序列产生的线程 900。

在线程 900 中，代码 901、910 和 913 为标号代码。

代码 902 到 909 对应于没有任何指向其他执行路径的分支的程序块 500 和 504。

在控制选择该执行路径时，代码 911 和 912 停止其他线程 700 和 800。

代码 914 和 915 相当于线程 900 的结束操作。

图 7、8 和 9 分别示出的代码行 702、802 和 902 基本一样的代码，上述代码表示在一个寄存器中存储 a 的，但是指定不同的寄存器。原因在于该目标硬件 130 为存储器共享型并且因此如果将 a 存储于同一寄存器中，则不能保证在各线程中数值的一致性，这样便不能产生编程人员所需的执行结果。

图 10 所示为包括用于使目标硬件 130 并行执行图 6 到图 9 所示的线程 600、700、800 和 900 的线程控制代码的线程 1000。当目标硬件 130 为存储器共享型的情况线程 1000 为主线程。

在线程 1000 中，代码 1001 到 1004 设置对应于根据分析信息 104 和执行频率信息 140 指定的频繁执行路径的线程。在该实施例中，假设目标硬件 130 具有足够的处理器内核数量，设定该对应于源程序段 510 的所有执行路径的线程。

由标号代码 1005 指定的代码 1006 到 1008 使得处理器内核启动相应的线程。

由标号代码 1009 指定的代码 1010 到 1012 等待相应的线程结束。

由标号代码 1013 指定的代码 1014 到 1016 放弃相应的线程并在所有线程结束后释放处理器内核。

编译器器件 100 产生包括主线程 1000 和线程 600、700、800 和 900 的可执行程序。这里注意线程 600、700、800 和 900 并行执行。

以下对图 6 到 14 以及 21 所示的代码进行说明。

如上所述，图 6 示出没有优化基本上由源程序段 510 直接转换来的代码。

在目标硬件 130 为存储器共享型的情况，图 7、8 和 9 分别示出通过对执行路径 551、执行路径 552 和连接程序块 501 和 504 的执行路径进行优化产生的代码，并且图 10 示出线程控制代码。另一方面，目标硬件 130 为存储器分布型的情况，图 12、13 和 14 分别示出通过对执行路径 551、执行路径 552 和连接程序块 501 和 504 的执行路径进行优化产生的代码，并且图 21 示出线程控制代码。

此外，图 10 示出由目标硬件 130 并行执行的可执行指令数量已知的情况下的线程控制代码，而图 11 示出由目标硬件 130 并行执行的可执行指令数量未知的情况下的线程控制代码。

在以下的说明中，每个地址表示在处理器中的指令地址，例如寄存器或者存储在寄存器中数值的地址。

代码“mov（address 1），（address 2）”表示在位于 address 2 的寄存器中存储 address 1 中的数值。例如，图 6 中的代码 602 在寄存器 D0 中存储位于地址 a 中的数值。

代码“add（address 1），（address 2）”将 address 1 中的数值和 address 2 中的数值加和并采用产生的和更新 address 2 中的数值。例如，图 6 的代码 604 将寄存器 D1 的数值和寄存器 D0 的数值加和并将由此产生的和存储在寄存器 D0 中。

代码“sub（address 1），（address 2）”从 address 2 中的数值中减去 address 1 中的数值和并采用产生的差值更新 address 2 中的数值。例如，图 6 的代码 612 从寄存器 D0 的数值中减去寄存器 D1 的数值并将由此产生的差值存储在寄存器 D0 中。

代码“cmp（address 1），（address 2）”将 address 1 中的数值和 address 2 中的数值进行比较。例如，图 6 的代码 606 将 0 和寄存器 D0 的值进行比较。

代码“bge（address 3）”表示如果在前一代码“cmp（address 1），（address 2）”中 address 2 中的数值不小于 address 1 的数值则跳转到 address 3 的代码。否则，控制进行紧随其后的代码。例如，如果在前述的代码 606 中寄存器 D0 中的数值不小于 0，则图 6 中的代码 607 使得不执行代码 608 而跳转到代码 609。

如果在前述代码“cmp（address 1），（address 2）”中 address 2 中的数值小于 address 1 中的数值则代码“blt（address 3）”跳转到 address 3 的代码。否则，

控制进行紧随其后的代码。例如，如果在前述的代码 705 中寄存器 D10 中的数值小于 0，则图 7 中的代码 706 使得在跳过代码 707 到 715 的同时跳转到代码 716。

代码“jmp（address 1）”跳转到 address 1 的代码。例如，图 6 中的代码 608 跳转到代码 627 同时跳过 609 到 626。

代码“not（address 1）”表示对 address 1 中数值的每一位进行取反，即 address1 的补码形式，并采用由此产生的值更新 address1 中的值。例如，图 6 的代码 629 对寄存器 D0 的每一位取反（补码形式）并在寄存器 D0 存储产生的值。

代码“inc（address 1）”表示对 address 1 中的数值加 1，并采用由此产生的和更新 address 1 中的数值。例如，图 6 的代码 630 对寄存器 D0 的数值加 1 并在寄存器 D0 中存储由此产生的和。

代码“dec（address 1）”表示对 address 1 中的数值减 1，并采用由此产生的差更新 address 1 中的数值。例如，图 11 的代码 1113 对寄存器 D1 中的数值减 1 并在寄存器 D1 中存储由此产生的差。

代码“clr（address 1）”通过设定 address 1 中的值为 0 的方式对 address 1 中的数值进行清零。例如，图 6 中的代码 633 对寄存器 D0 的值进行清零以初始化寄存器 D0。

代码“as1（address 1），（address 2）”用于避免由目标硬件 130 所使用的指令字长度差异引起的地址不一致。该代码主要用于从一个代码向另一个代码转换。在指令字长单元中管理程序中每个指令的地址。假设该指令字长为 8 位。如果指令 1 的地址是 0，则在指令 1 后的指令 2 的地址是 8。在从指令 1 向指令 2 转换时，仅仅对指令 1 的地址加 1 不可能产生指令 2 的地址，并因此由于地址的不一致导致不能执行指令 2。因此，代码“as1（address 1），（address 2）”将 address 2 中的数值与表示指令字长的 address 1 中的数值相乘，并在 address 2 的寄存器中存储由此产生的乘积。

代码“ret”使得返回到主线程。

以下说明线程控制代码。

代码“_createthread（address 1），（address 2）”创造开始于 address1 的线程，并在位于 address 2 的寄存器中存储关于线程执行的信息。例如，图 10 的

代码 1002 创建开始于 LABEL500-501-502 的线程，即图 7 所示的线程 700，并在 THREAD500-501-502 中存储关于线程执行的信息。

代码“_beginthread（address）”表示在 address 开始线程。例如，图 10 的代码 1006 启动开始于 LABEL500-501-502 的线程，即图 7 所示的线程 700。

代码“_endthread”将线程设定为结束状态并返回表示线程结束的信息。例如，图 7 的代码 717 结束线程 700 并向主线程返回表示线程 700 结束的信息。

代码“_deletethread（address）”放弃开始于 address 的线程。例如，图 10 的代码 1014 放弃开始于 LABEL500-501-502 的线程，即图 7 所示的线程 700。

代码“_killthread（address）”停止执行开始于 address 的线程。例如，即使线程 800 还在执行， 图 7 的代码 714 停止开始于 LABEL500-501-503 的线程，即图 8 所示的线程 800。

代码“_waitthread（address ）”等待开始于 adress 的线程的完成。可以由上述的“_endthread”信息通知完成。例如，图 10 的代码 1010 等待 THREAD500-504 的完成，即图 9 所示的线程 900。

代码“_commit（address 1），（address 2）”将产生于任何主线程和其他线程的 address1 中的信息映射给包括主线程和其他线程的全体的位于 address 2 的寄存器。

代码“_broadcast（address 1），（address 2）”在目标硬件 130 为存储器分布型的情况向目标硬件中所有连接到处理器内核的存储器映射一个处理器内核的处理结果。该代码采用对应于处理器内核的存储器中 address 1 中的值更新所有存储器中 address 2 中的值。

代码“_getparrallelnum（address）”将由目标硬件 130 可并行执行的线程数量返回给 address。该代码用于检测目标硬件 130 中可以并行执行的处理器内核的数量。特别地，编译过程中当在目标硬件 130 中可以并行执行的处理器内核的数量为未知时该代码是必要的。

（操作）

以下采用流程图说明编译器器件 100 在产生可执行程序 120 的操作。

在编译器器件 100 中输入源程序 110 时，分析单元 101 获得源程序 110 中关于分支和循环的信息，基于该获得的信息检测执行路径，并分配执行路径的

标识符。

最初，通过优化单元 103 和代码转换单元 104 将源程序 110 不进行优化地转换为可执行程序。为了获得关于执行路径执行频率的信息，在目标硬件中 130 执行该可执行程序。

图 15 所示为获取关于执行路径执行频率信息的操作流程图。

为了测量源程序段 510 中执行路径的执行频率，该优化单元 103 不进行优化地转换源程序段 510 并插入分析代码(profiling code)从而产生可执行代码。代码转换单元 104 将可执行程序转换为可以在目标硬件 130 中运行的可执行程序（S1500）。这里提到的分析代码用于检测条件分支选择了哪个执行路径。无论何时只要控制选择了该执行路径该分析代码就在该执行路径对应的标志符上累加 1。在插入分析代码时，可执行程序的执行速度降低。因此，不会将分析代码插入由编译器器件 100 最终产生的预期可执行程序中。

然后，目标硬件 130 执行基本直接由源程序转换的可执行程序（S1502），以计算执行路径的执行频率。每次选择执行路径时，向对应于执行路径标识符的计数累加 1。以这种方式计算出的表示执行路径执行频率的信息作为执行频率信息 140 存储在目标硬件 130 的 RAM 中。然后将该执行频率信息 140 输出到编译器器件 100 中的执行路径指定单元 102。基于该信息，产生预期可执行程序。

当向编译器器件 100 输出执行频率信息 140 时，该目标硬件 130 还输出关于硬件配置的信息。该信息包括表示目标硬件 130 存储器类型的存储器信息以及表示在目标硬件 130 中可以并行执行的处理器内核数量的并行执行信息。这些信息事先存储在目标硬件 130 的 ROM 中，并和执行频率信息 140 一起输出给编译器器件 100。

图 19 所示为通过编译器器件 100 产生预期可执行程序的操作流程图。

首先，优化单元 103 产生基本上直接将源程序 110 转换为可执行形式的第一代码（S1901）。执行路径指定单元 102 基于从目标硬件 130 获得的执行路径频率信息 140 按执行频率降序的顺序提取一个或者多个优先执行路径，即一个或者多个频繁执行路径（S1905）。该优化单元 103 基于在目标硬件 130 中可以并行执行的处理器内核数量通过优化每个优先执行路径中的指令序列产生第二代码（S1907）。这里，可以产生对应于不同优先执行路径其中之一的多组第

二代码，第二代码的数量比可以并行执行的处理器内核数量少 1。详细地说，对于按执行频率降序排列的每个优先执行路径，产生对应于优化后的这些执行路径中指令的线程。作为一个实施例，如果可以并行执行的处理器内核的数量为 4，则产生具有对应于第一到第三高执行频率的线程。这里注意第一代码和用于控制已产生的多组第二代码的代码包括在同一线程中。

此后，该代码转换单元 104 产生适用于目标硬件 130 的可执行程序，通过该组织后的代码并行执行第一代码和多组第二代码（S1909）。

以下采用将图 5A 所示的源程序段转换为可执行程序的具体实施例来详细说明该操作。

当在编译器器件 100 中输入包括源程序段 510 的源程序 110 时，分析单元 101 分析该源程序段 510，并检测出三个执行路径，执行路径 500-501-502（执行路径 551）、执行路径 500-501-503（执行路径 552）和如图 5B 所示的执行路径 500-504。分析单元 101 对于每个执行路径分配识别码。优化单元 103 产生不经过优化基本直接将源程序段 510 转换为编码码的线程 600 的代码。优化单元 103 在产生的代码中插入分析代码。代码转换单元 104 将该代码转换为适用于目标硬件 130 的可执行程序。

目标硬件 130 执行可执行程序。基于该执行，目标硬件 130 产生表示执行路径执行频率的执行频率信息 140，并将该信息输出给编译器器件 100。例如，执行频率信息 140 显示执行路径 500-501-502 已经执行了 24 次，执行路径 500-501-503 已经执行了 15 次，并且执行路径 500-504 已经执行了 3 次。目标硬件 130 还向编译器器件 100 输出关于其硬件配置的信息。例如，该信息包括设定表示存储器共享型为 0 的存储器信息以及表示可以并行执行的处理器内核的数量为 4 的并行执行信息。

该执行路径指定单元 102 接收执行频率信息 140。基于该执行频率信息 140，优化单元 103 产生主线程 1000。由于可以并行执行的处理器内核的数量为 4，因此并发可执行线程的数量为包括包含在主线程 1000 中的线程 600 的四个线程。因此，在主线程 1000 中产生三个线程 700、800 和 900。优化单元 103 产生用于使每个线程通过单独处理器内核执行的代码。代码转换单元 104 通过由优化单元 103 产生的代码产生适用于目标硬件 130 的可执行程序 120。

以上说明使用了显然跟随有另一源程序段的源程序段 510 的实施例。如果

任何线程 700、800 和 900 的执行条件为真，则在那个线程后执行对应于下一源程序段的可执行代码。如果每个线程 700、800 和 900 的执行条件为假，则在线程 600 后执行对应于下一源程序段的可执行代码。

第二实施方式

本发明的第二实施方式说明目标硬件 130 为存储器分布型的情况。以下说明主要集中于和第一实施方式的不同之处。

第二实施方式和第一实施方式的不同点在于，由于每个处理器内核均连接到单独的存储器上并使用连接的存储器上的值，因此不同于存储器共享型的情况，不会存在由存储器访问竞争引起的性能降低的危险。

采用图 12 到 14 和 21 所示的代码对此进行详细说明。图 12 示出和图 7 所示的线程 700 具有同样执行内容的线程 1200。图 13 示出和图 8 所示的线程 800 具有同样执行内容的线程 1300。图 14 示出和图 9 所示的线程 900 具有同样执行内容的线程 1400。图 21 示出存储器分布型情况下的主线程 2100。

在目标硬件 130 为存储器共享型时，数值 a 需要存储在每个线程 700、800 和 900 的寄存器中，如图 7 到图 9 中由代码 702、802 和 902 所示。在存储器分布型中，这种存储是不必要的，由于主线程 2100 向对应于图 21 所示的代码 2104 到 2106 所示的线程 1200、1300 和 1400 的存储器的寄存器传送数值 a。

更详细地，代码 2105 使对应于由代码 2101 到 2103 产生的线程 1200、1300 和 1400 的处理器内核以在各自存储器的寄存器 D0 中存储数值 a。

同样，代码 2106 使对应于有由代码 2101 到 2103 产生的线程 1200、1300 和 1400 的处理器内核以在各自存储器的寄存器 D1 中存储数值 b。

如果任何线程 1200、1300 和 1400 的执行条件为真，则该线程的执行结果需要映射给连接到运行主线程 2100 的处理器内核的存储器中。这可以通过“_commit”代码实现。例如，图 12 所示的代码 1215 和代码 1216 为这种代码。该代码使得线程的执行结果映射到主线程的存储器中。

在目标硬件 130 为存储器分布型的情况，编译器器件 100 产生组织为包括线程 1200、1300 和 1400 以及包含线程 600 的主线程 2100 的可执行程序。这种可执行程序可以在目标硬件 130 中正确地执行同时维持数值一致性。

以下通过参照图 17 的流程图，说明在存储器分布型的情况可执行程序的

步骤。下面的说明主要集中于主线程2100的步骤。

首先，产生要通过其他处理器内核执行的线程，即线程1200、1300和1400（S1700）。将在前一源程序段中获得的数据传送到并存储在这些处理器内核每个的存储器中（S1701）。此后，执行每个线程（S1702）。一旦所有线程已经结束（S1703），终止该线程（S1704）。

第三实施方式

第一和第二实施方式说明了对于编译器器件100目标硬件130可以并行执行的指令数量为已知的情况。但是，还可能存在在目标硬件130中可以并行执行的指令数量为未知的情况。该情况包括当事先将执行频率信息140和存储器信息提供给编译器器件100时，该编译器器件100需要在没有从目标硬件130向编译器器件100的信息传送的情况下产生可执行程序120。在这种情况，在该主线程中需要包含用于获取处理器内核数量的代码和用于根据处理器内核的数量设定线程数量的代码。图11示出处理器内核数量为未知情况下主线程1100的代码。以下说明该代码的执行内容。这里假设该编译器器件100产生如图6到9的四个线程600、700、800和900。

有标号代码1104指定的代码1105到1117获取目标硬件130的处理器内核的数量并根据该处理器内核的数量设定线程的数量。

首先，获取由该编译器器件100产生的线程数量，由m表示，并存储在寄存器D0中（代码1105）。接下来，获取可以在目标硬件130中并行执行的处理器内核的数量，由n表示，并存储在寄存器D1中（代码1106）。将寄存器D0中的数量m和寄存器D1中的数量n进行比较（代码1107）。如果$n \geqq m$，该控制跳转到标号代码1110（代码1108）。如果$n<m$，控制跳转到标号代码1112（代码1109）。

如果$n \geqq m$，没有必要进行调节，因此将m存储在寄存器D1中（代码1111）。

如果$n<m$，线程数量超过了并发可执行指令的数量，这意味着不可能执行全部的指令。

因此，在寄存器D1中存储从寄存器D1中的数值n减1所获得的数量（代码1113）。该数量n-1表示可执行线程的数量。使用一个额外的处理器内核执行基本直接转换源程序110得到的线程600。

接下来，为了计算指令地址，将 n-1 乘以指令字长（代码 1114）。例如，如果指令字长为 8 位，那么 n-1 乘以 8。此后，在寄存器 D2 中存储 P_POINTER（代码 1115）。从寄存器 D2 中的数值中减去寄存器 D1 中的数值，并且采用由此产生的差更新寄存器 D2（代码 1116）。此后，控制跳转到寄存器 D2 中的地址（代码 1117）。因此，寄存器 D2 中的数值确定要启动线程 700、800 和 900 中的哪个线程。例如，如果可以并行执行处理器内核的数量为 2，控制跳转到代码 1121。如果可以并行执行处理器内核的数量为 3，控制跳转到代码 1120。这里注意代码 1119 到代码 1121 分别启动对应于按执行频率升序排列的执行路径的线程 900、800 和 700。

通过采用这种主线程 1100，该编译器器件 100 即使在在目标硬件 130 中可以并行执行的指令数量为未知的情况下也可以产生预期可执行程序 120，尽管在图 11 中省略，但是代码 1126 以后的代码和图 10 中的代码 1012 以后的代码一样。

图 16 所示为对目标硬件 130 的硬件配置进行判断操作的流程图。

首先，优化单元 103 判断在目标硬件 130 中可以并行执行的指令数量为已知还是未知（S1601）。根据编译器器件 100 是否已经从目标硬件 130 获得并行执行信息可以进行判断。如果并发执行的线程数量为未知，则产生图 11 所示的代码。优化单元 103 还获取存储器信息，并判断该目标硬件 130 为存储器共享型还是存储器分布型（S1603）。基于该判断，产生可执行程序 120。

第四实施方式

本发明的第四实施方式和第一到第三实施方式不同点在于用于执行程序的单元包括在编译器器件中。图 18 所示为已经包括用于执行程序的单元的程

序转换和执行器件 1800 的框图。

更详细地，除了编译器器件 100 的结构部件外，该程序转换和执行器件 1800 包括源程序存储单元 1801、可执行程序存储单元 1806，以及执行单元 1807。这省去了为了使目标硬件执行原始可执行程序以获取执行频率信息而连接到目标硬件的麻烦。该程序转换和执行器件 1800 可以获取可执行程序的执行结果和其本身的执行频率信息。

源程序存储单元 1801 存储输入的源程序。

可执行程序存储单元 1806 用于存储由代码转换单元 1805 产生的可执行程序。该可执行程序存储单元 1806 包括 RAM。

执行单元 1807 从可执行程序存储单元 1806 中读取可执行程序，并执行该读取的可执行程序。执行单元 1807 包括 MPU、ROM 和 RAM 并以和图 1 所示的目标硬件 130 同样的方式实现功能。执行单元 1807 的 MPU 由多个处理器内核构成。

产生于程序转换和执行器件 1800 中的代码和第一到第三实施方式中的一样。

根据该结构，程序转换和执行器件 1800 可以用作在转换程序同时执行程序的解释程序（interpreter）。

变型实施例

尽管已经通过上述实施方式说明了本发明，但是本发明不限于以上陈述。以下给出实施例的变型。

（1）第一和第二实施方式说明了目标硬件具有足够数量的用于执行所有产生线程的处理器内核的情况。如果仅有几个处理器内核，例如 2 个，但是，组织该主线程使得仅并行执行线程 600 和线程 700。在该情况下，省略图 10 所示的代码 1003、1004、1007、1008、1011、1012、1015 和 1016。

（2）上述实施方式说明了假设第一代码，即如 3 所示的线程 300 比其他线程慢，产生预期可执行程序。另外，考虑到线程 300 比其他线程快的情况，用于停止其他线程的代码可以插入线程 300 的末端。

（3）上述的实施方式说明了目标硬件具有多个处理器内核的情况。作为替代，把一台个人计算机作为一个处理器内核，多台个人计算机通过网络连接

到编译器器件上以执行并行执行。

（4）上述实施方式说明了一个线程的执行条件为真的情况，执行另一线程的处理器内核停止执行，删除线程和操作数据，并然后执行新分配的线程。然而，当同一线程一次又一次的循环，每次重新分配同一线程是低效率的，这可能会降低目标程序的执行速度。因此，如果下一线程和当前线程一样并且仅仅操作数据不同，则可以产生包括用于保留当前线程不抛弃该线程并且仅传送必要的操作数据的目标程序。

（5）上述实施方式说明了通过彼此连接操作的器件的功能单元产生源程序的情况。但是，本发明还可以用于通过根据上述操作步骤产生目标程序的方法来实现。

尽管已经参照附图以实施例的方式对本发明进行了充分描述，但是应该注意的是，很显然对于本领域的技术人员而言，可以对本发明做出各种变型和改进。

因此，除非这些变型和改进脱离开本发明的范围，否则将认为它们包含在本发明中。

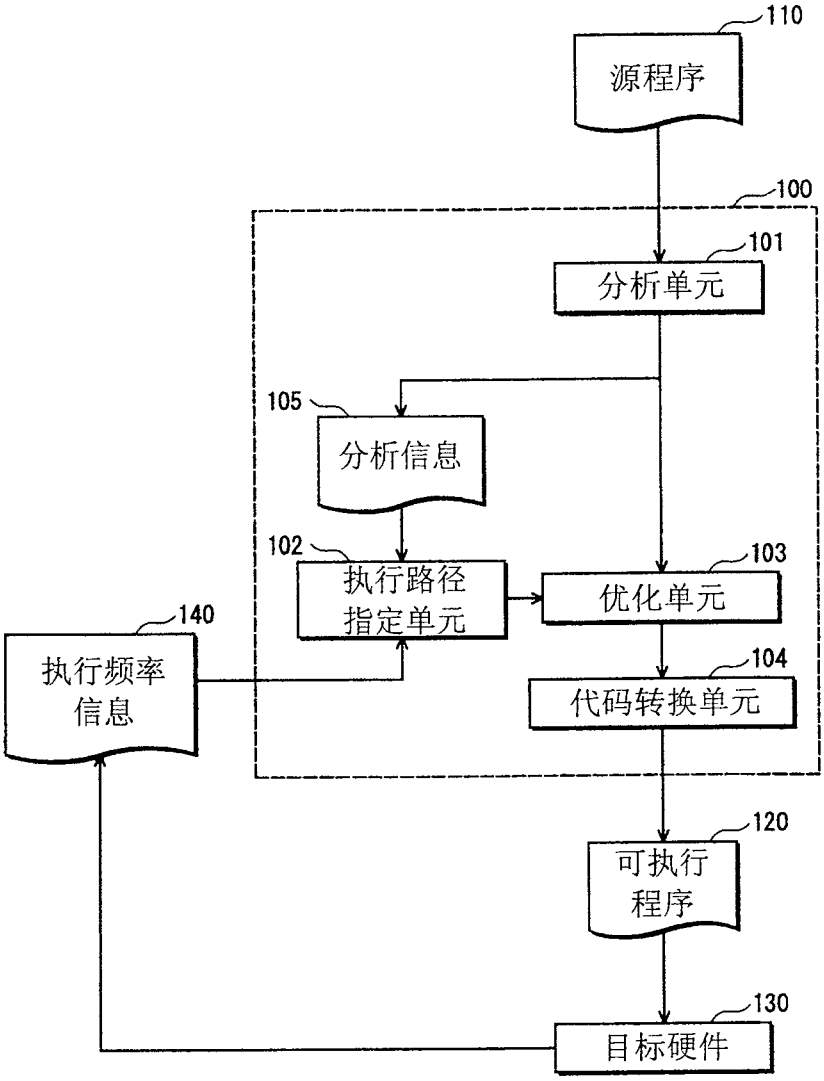

110
源程序
100
101
分析单元
105
分析信息
102
执行路径
指定单元
103
优化单元
140
执行频率
信息
104
代码转换单元
120
可执行
程序
130
目标硬件

图 1

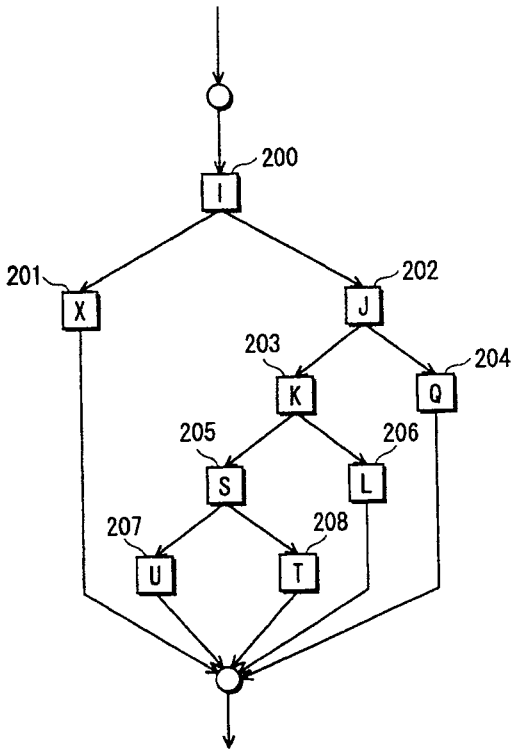


图 2

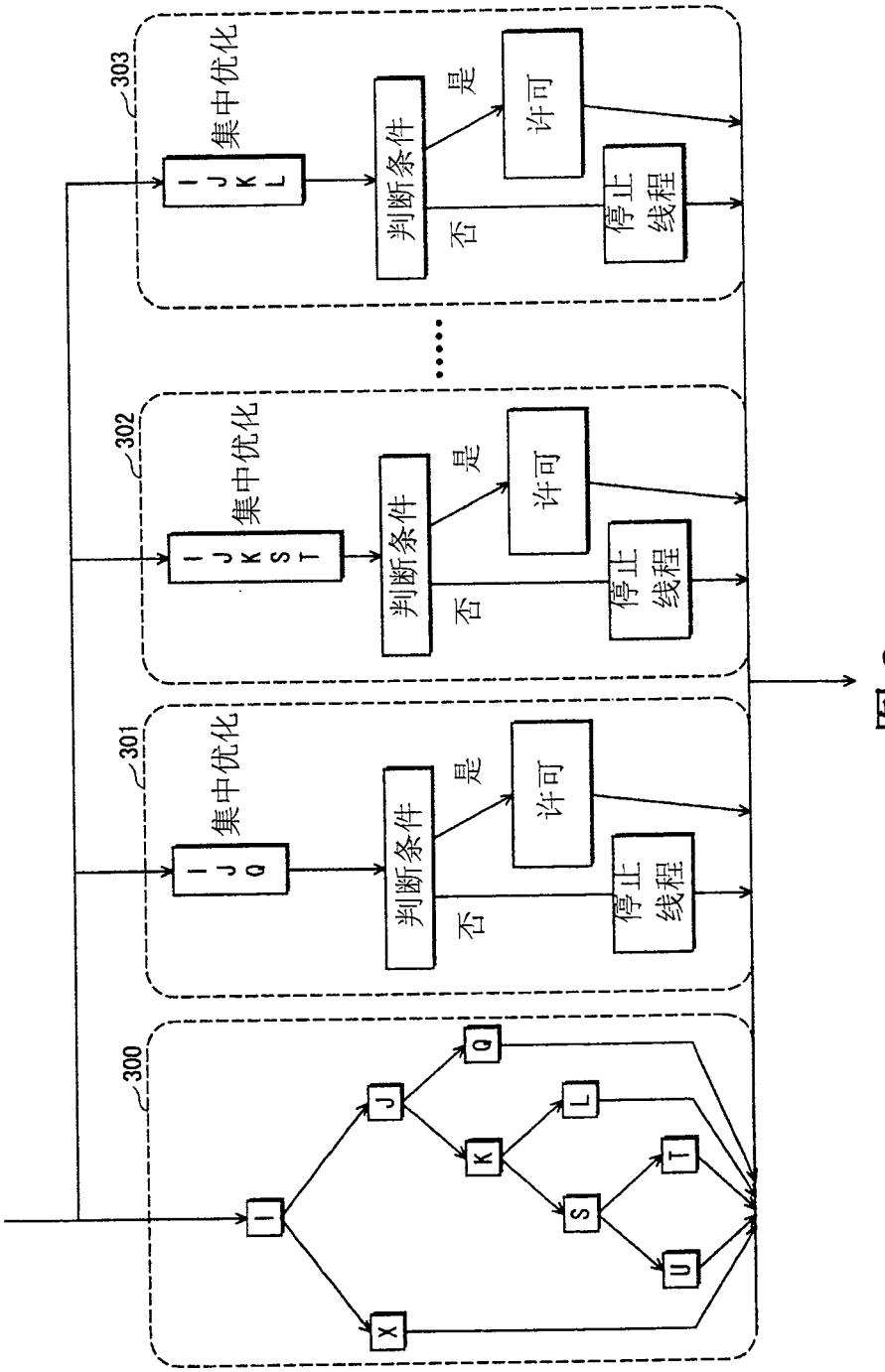

300
I
J
K
L
S
T
U
Q
X
301
集中优化
I J Q
判断条件
是
否
许可
停止线程
302
集中优化
I J K S T
判断条件
是
否
许可
停止线程
303
集中优化
I J K L
判断条件
是
否
许可
停止线程

图 3

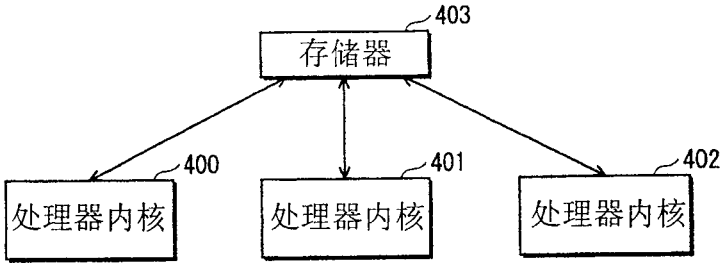

403
存储器
400
处理器内核
401
处理器内核
402
处理器内核

图 4A

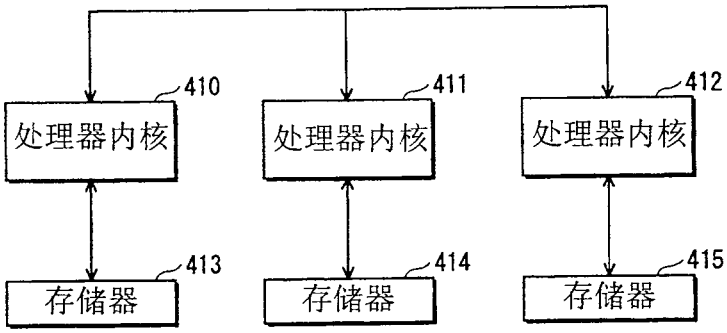

410
处理器内核
411
处理器内核
412
处理器内核
413
存储器
414
存储器
415
存储器

图 4B

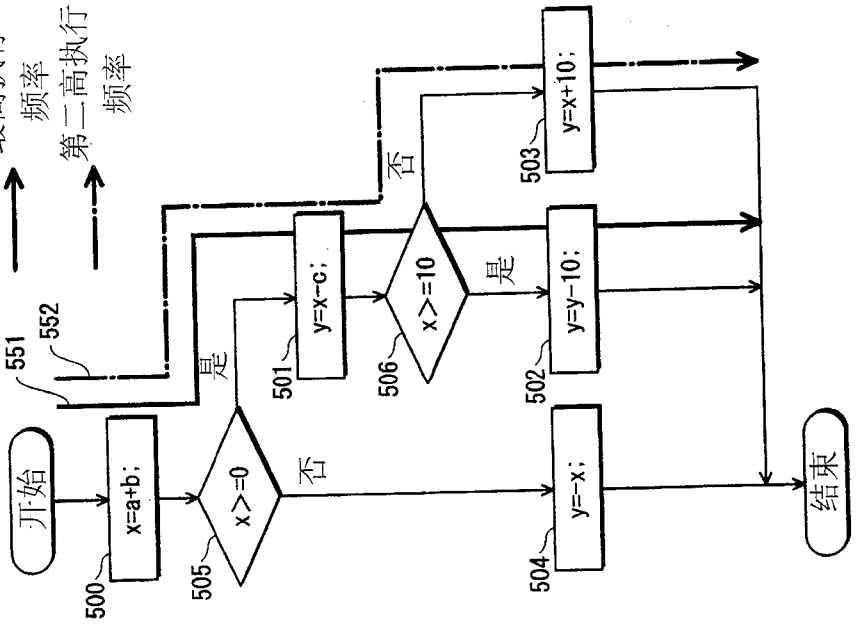

最高执行频率
第二高执行频率
551
552
开始
x=a+b;
500
x>=0
505
是
否
y=x-c;
501
x>=10
506
是
否
y=y-10;
502
y=x+10;
503
y=-x;
504
结束

图 5B

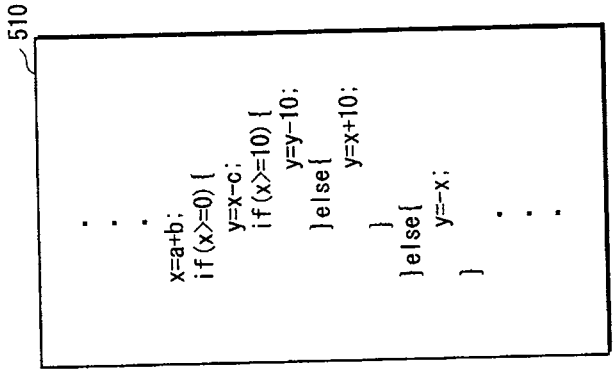

510
x=a+b;
if(x>=0){
y=x-c;
if(x>=10){
y=y-10;
}else{
y=x+10;
}
}else{
y=-x;
}

图 5A

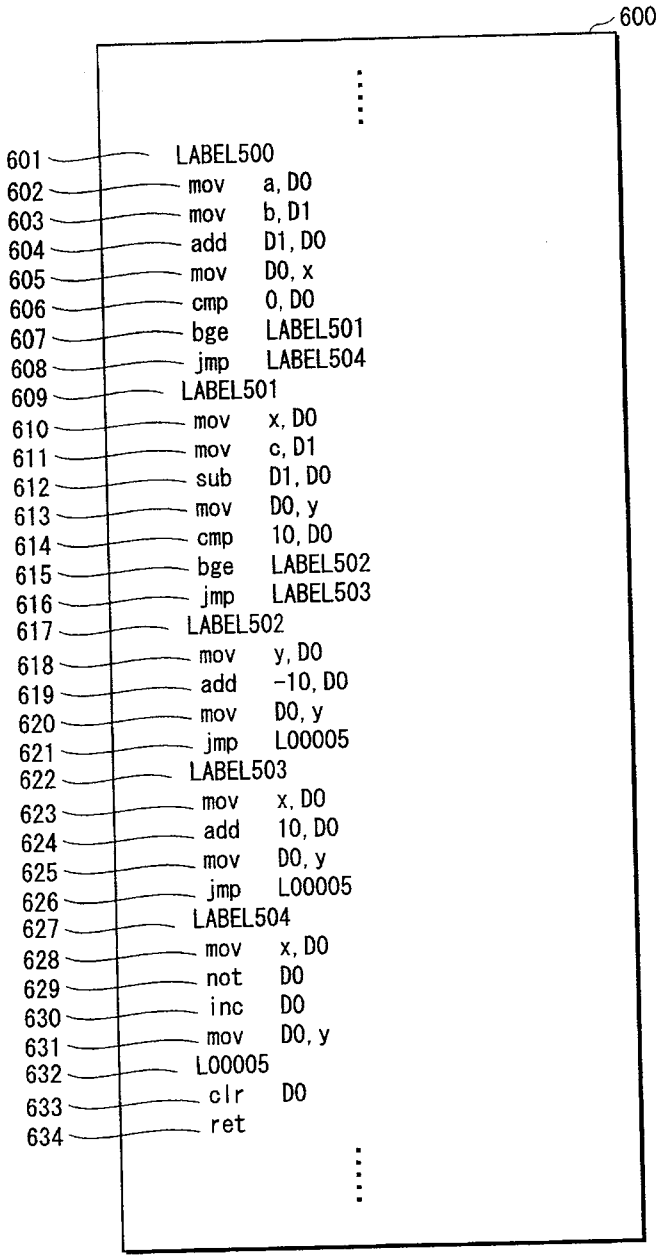

600
601 LABEL500
602 mov a, D0
603 mov b, D1
604 add D1, D0
605 mov D0, x
606 cmp 0, D0
607 bge LABEL501
608 jmp LABEL504
609 LABEL501
610 mov x, D0
611 mov c, D1
612 sub D1, D0
613 mov D0, y
614 cmp 10, D0
615 bge LABEL502
616 jmp LABEL503
617 LABEL502
618 mov y, D0
619 add -10, D0
620 mov D0, y
621 jmp L00005
622 LABEL503
623 mov x, D0
624 add 10, D0
625 mov D0, y
626 jmp L00005
627 LABEL504
628 mov x, D0
629 not D0
630 inc D0
631 mov D0, y
632 L00005
633 clr D0
634 ret

图 6

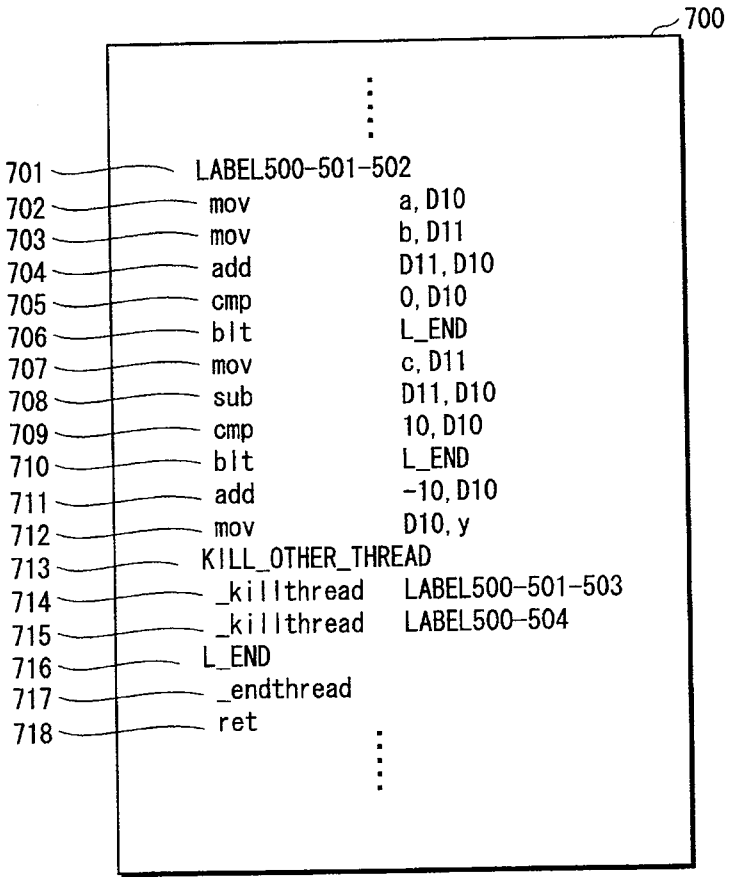

700
701 LABEL500-501-502
702 mov a, D10
703 mov b, D11
704 add D11, D10
705 cmp 0, D10
706 blt L_END
707 mov c, D11
708 sub D11, D10
709 cmp 10, D10
710 blt L_END
711 add -10, D10
712 mov D10, y
713 KILL_OTHER_THREAD
714 _killthread LABEL500-501-503
715 _killthread LABEL500-504
716 L_END
717 _endthread
718 ret

图 7

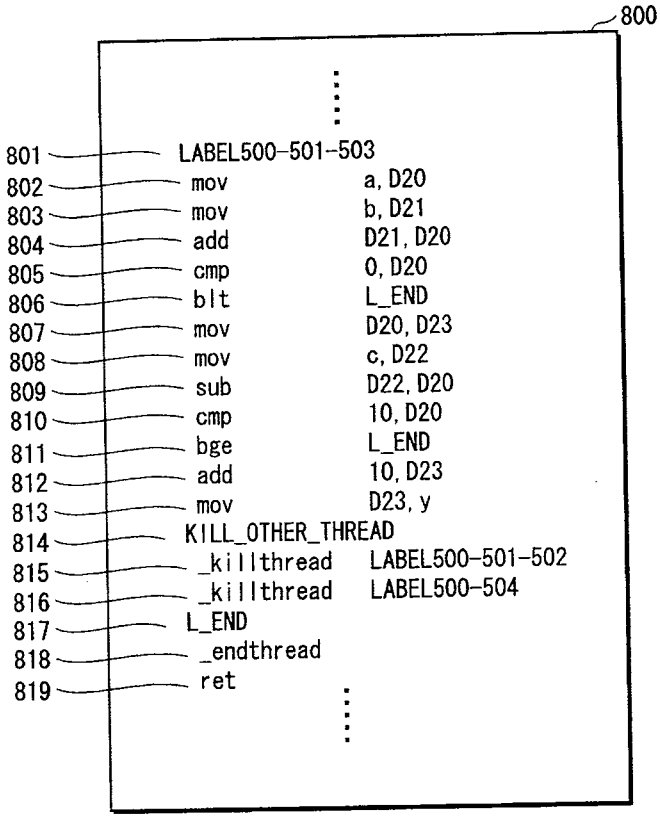

800
801 LABEL500-501-503
802 mov a, D20
803 mov b, D21
804 add D21, D20
805 cmp 0, D20
806 blt L_END
807 mov D20, D23
808 mov c, D22
809 sub D22, D20
810 cmp 10, D20
811 bge L_END
812 add 10, D23
813 mov D23, y
814 KILL_OTHER_THREAD
815 _killthread LABEL500-501-502
816 _killthread LABEL500-504
817 L_END
818 _endthread
819 ret

图 8

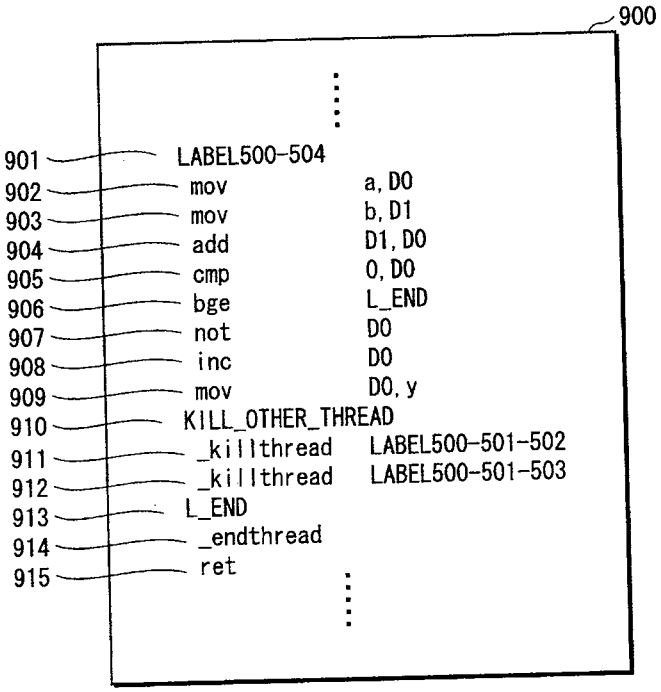

900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
LABEL500-504
mov a, D0
mov b, D1
add D1, D0
cmp 0, D0
bge L_END
not D0
inc D0
mov D0, y
KILL_OTHER_THREAD
_killthread LABEL500-501-502
_killthread LABEL500-501-503
L_END
_endthread
ret

图 9

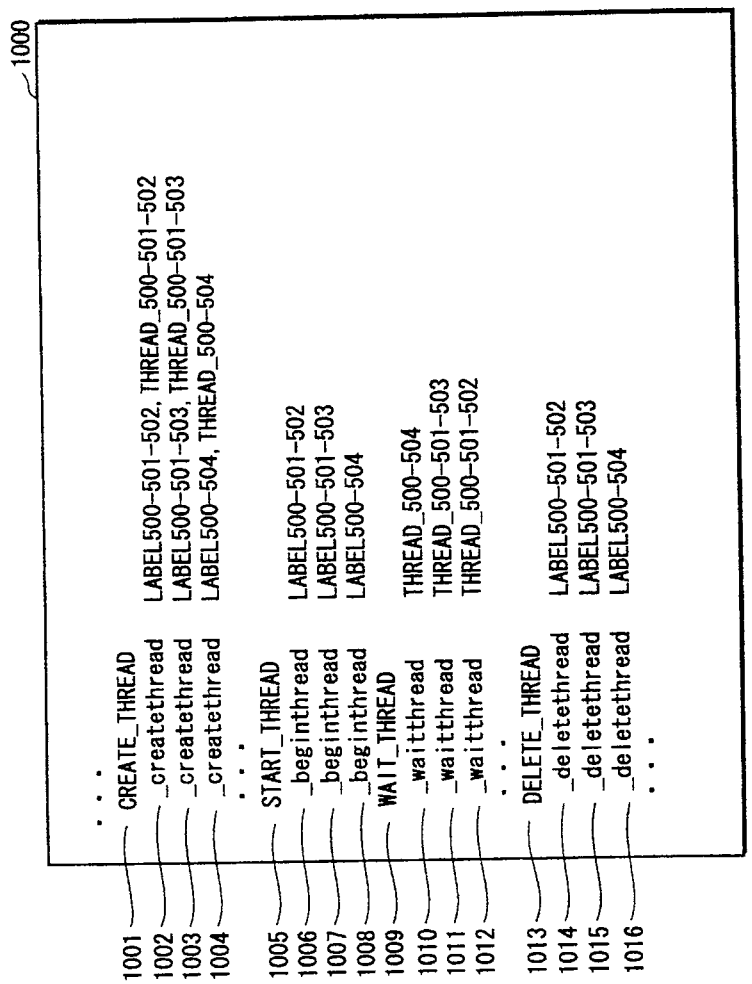

1000
. . .
CREATE_THREAD LABEL500-501-502, THREAD_500-501-502
_createthread LABEL500-501-503, THREAD_500-501-503
_createthread LABEL500-504, THREAD_500-504
. . .
START_THREAD
_beginthread LABEL500-501-502
_beginthread LABEL500-501-503
_beginthread LABEL500-504
WAIT_THREAD
_waitthread THREAD_500-504
_waitthread THREAD_500-501-503
_waitthread THREAD_500-501-502
. . .
DELETE_THREAD
_deletethread LABEL500-501-502
_deletethread LABEL500-501-503
_deletethread LABEL500-504
. . .
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016

图 10

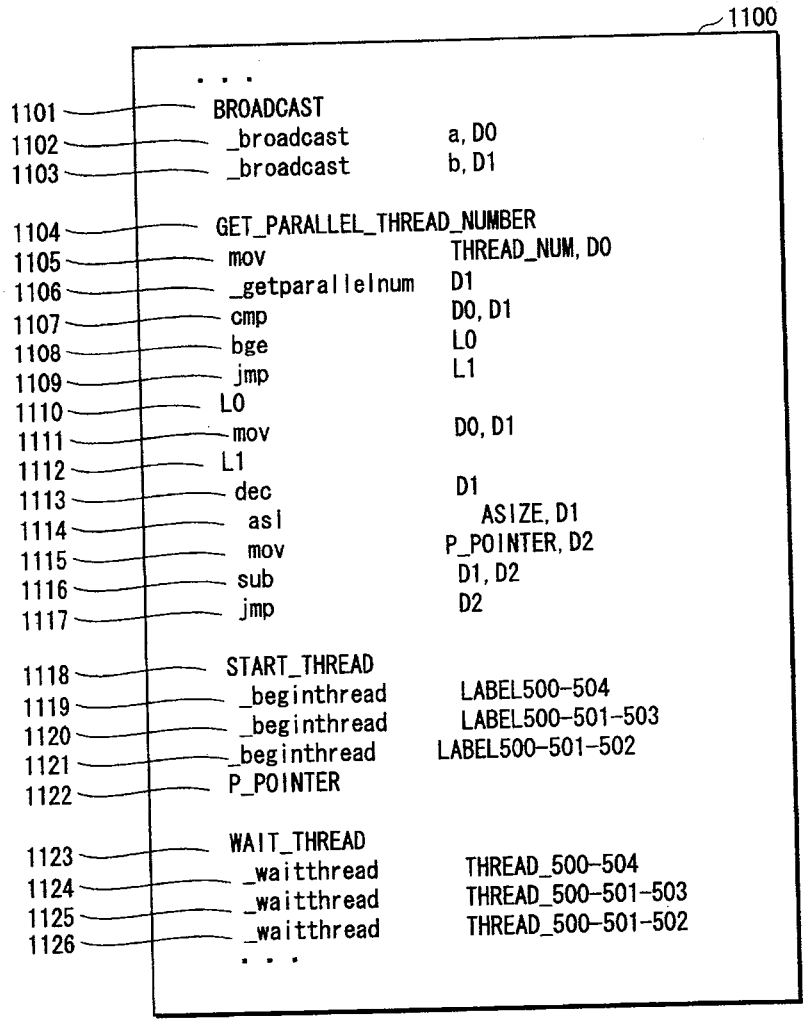

1100
. . .
1101 BROADCAST
1102 _broadcast a, D0
1103 _broadcast b, D1
1104 GET_PARALLEL_THREAD_NUMBER
1105 mov THREAD_NUM, D0
1106 _getparallelnum D1
1107 cmp D0, D1
1108 bge L0
1109 jmp L1
1110 L0
1111 mov D0, D1
1112 L1
1113 dec D1
1114 asl ASIZE, D1
1115 mov P_POINTER, D2
1116 sub D1, D2
1117 jmp D2
1118 START_THREAD
1119 _beginthread LABEL500-504
1120 _beginthread LABEL500-501-503
1121 _beginthread LABEL500-501-502
1122 P_POINTER
1123 WAIT_THREAD
1124 _waitthread THREAD_500-504
1125 _waitthread THREAD_500-501-503
1126 _waitthread THREAD_500-501-502
. . .

图 11

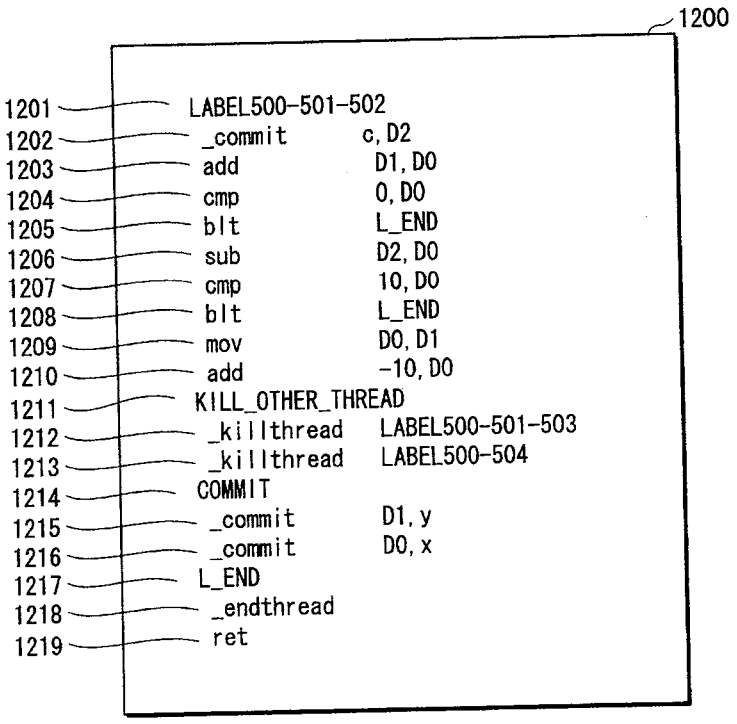

1200
1201 LABEL500-501-502
1202 _commit c,D2
1203 add D1,D0
1204 cmp 0,D0
1205 blt L_END
1206 sub D2,D0
1207 cmp 10,D0
1208 blt L_END
1209 mov D0,D1
1210 add -10,D0
1211 KILL_OTHER_THREAD
1212 _killthread LABEL500-501-503
1213 _killthread LABEL500-504
1214 COMMIT
1215 _commit D1,y
1216 _commit D0,x
1217 L_END
1218 _endthread
1219 ret

图 12

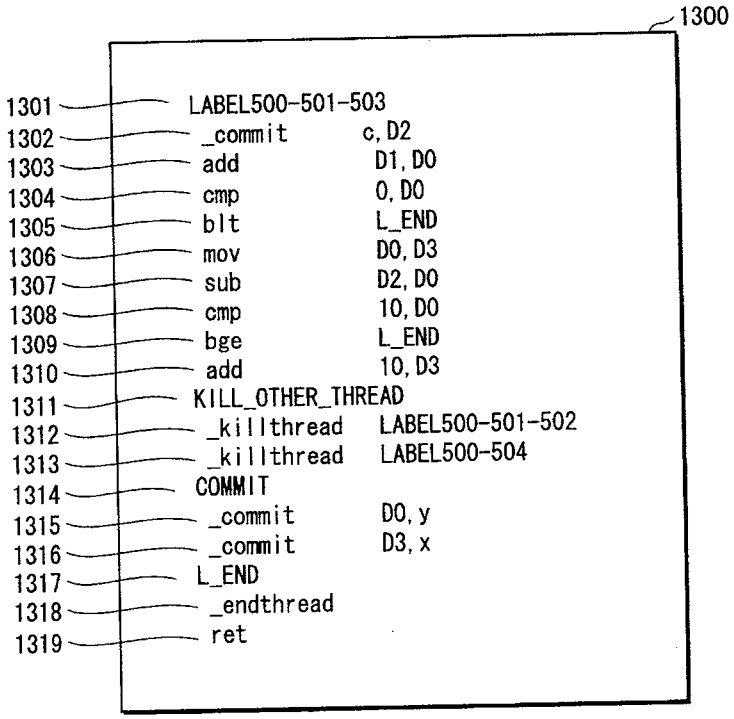

1300
1301 LABEL500-501-503
1302 _commit c, D2
1303 add D1, D0
1304 cmp 0, D0
1305 blt L_END
1306 mov D0, D3
1307 sub D2, D0
1308 cmp 10, D0
1309 bge L_END
1310 add 10, D3
1311 KILL_OTHER_THREAD
1312 _killthread LABEL500-501-502
1313 _killthread LABEL500-504
1314 COMMIT
1315 _commit D0, y
1316 _commit D3, x
1317 L_END
1318 _endthread
1319 ret

图 13

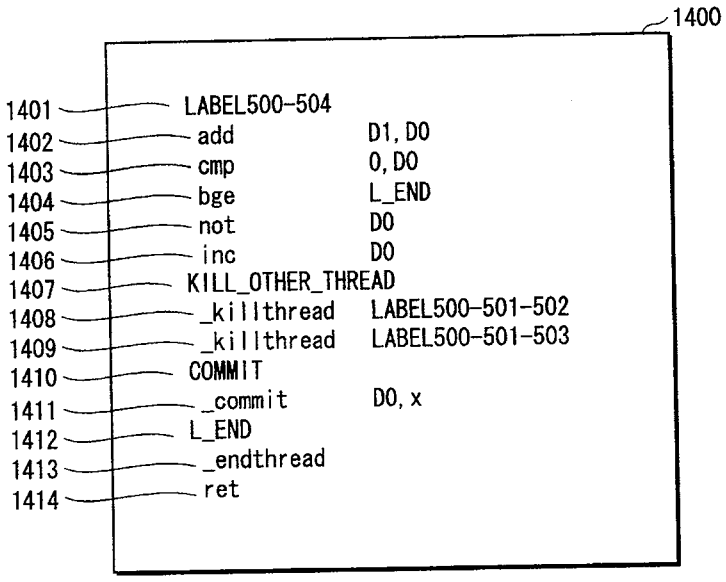

1400
1401
LABEL500-504
1402
add D1,D0
1403
cmp 0,D0
1404
bge L_END
1405
not D0
1406
inc D0
1407
KILL_OTHER_THREAD
1408
_killthread LABEL500-501-502
1409
_killthread LABEL500-501-503
1410
COMMIT
1411
_commit D0,x
1412
L_END
1413
_endthread
1414
ret

图 14

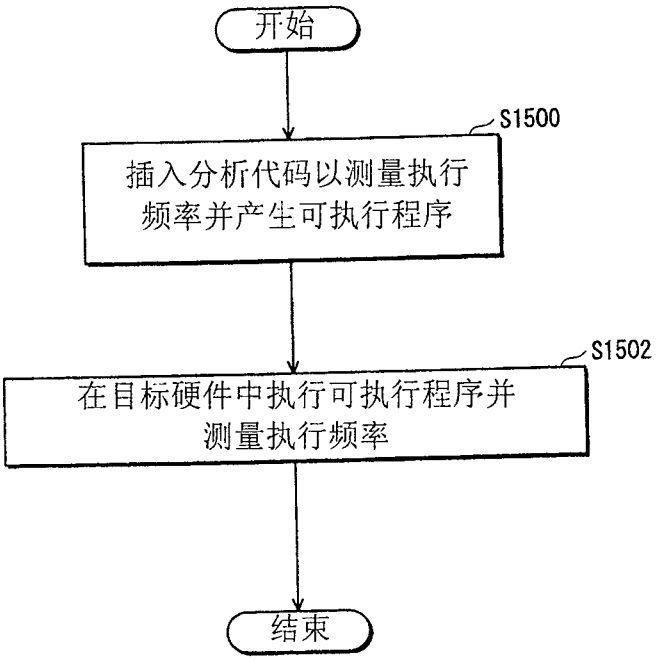

开始
S1500
插入分析代码以测量执行频率并产生可执行程序
S1502
在目标硬件中执行可执行程序并测量执行频率
结束

图 15

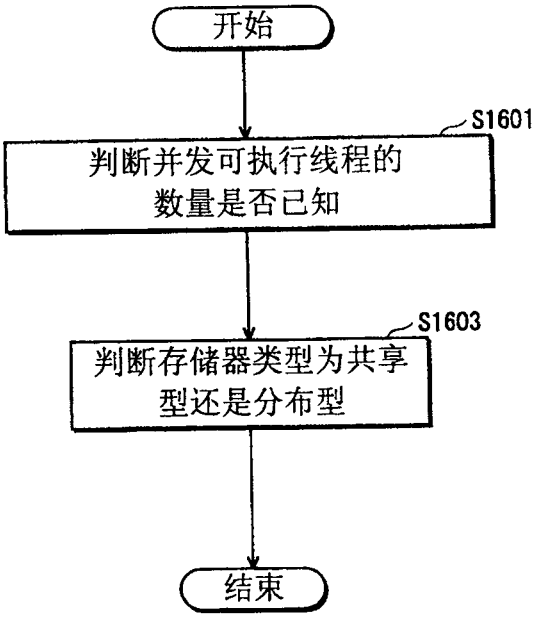


图 16

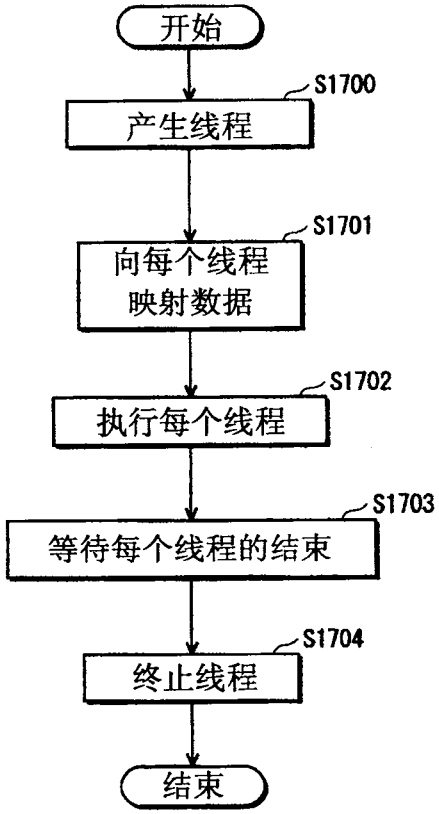

开始
S1700
产生线程
S1701
向每个线程
映射数据
S1702
执行每个线程
S1703
等待每个线程的结束
S1704
终止线程
结束

图 17

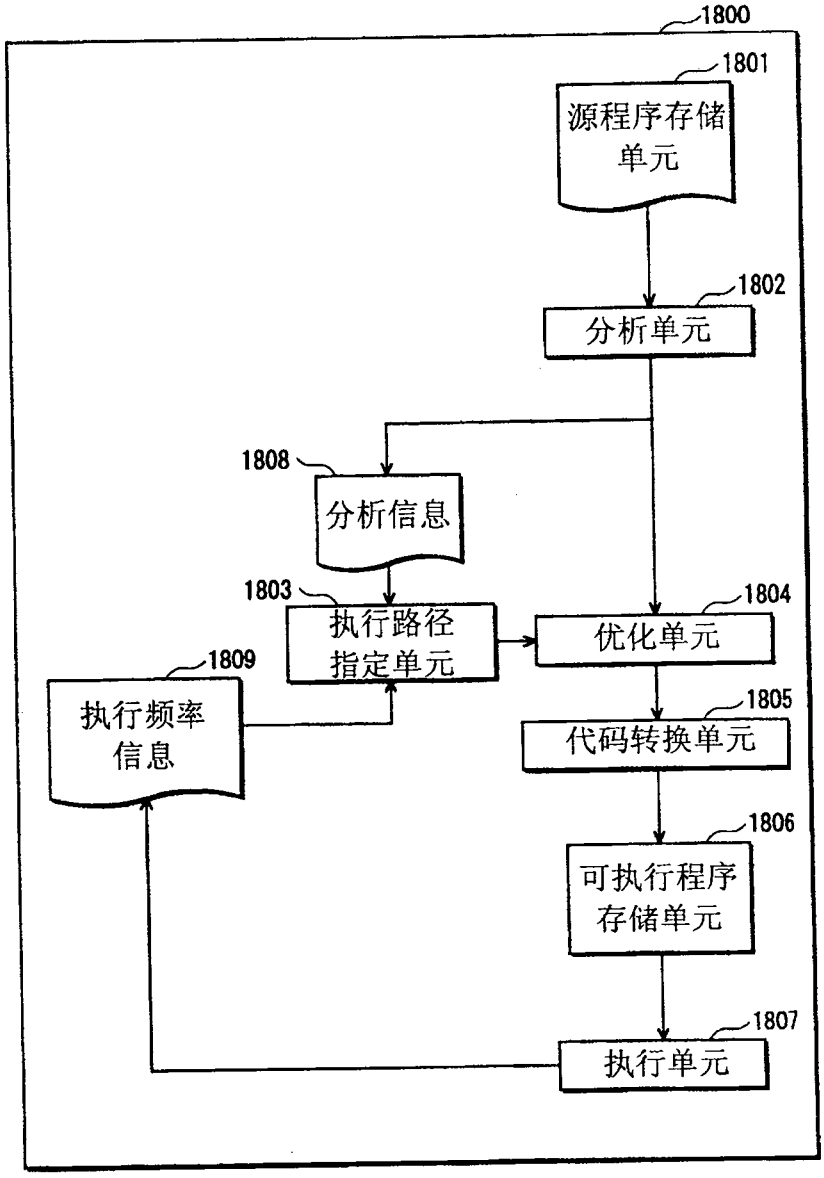

1800
1801
源程序存储单元
1802
分析单元
1808
分析信息
1803
执行路径指定单元
1804
优化单元
1809
执行频率信息
1805
代码转换单元
1806
可执行程序存储单元
1807
执行单元

图 18

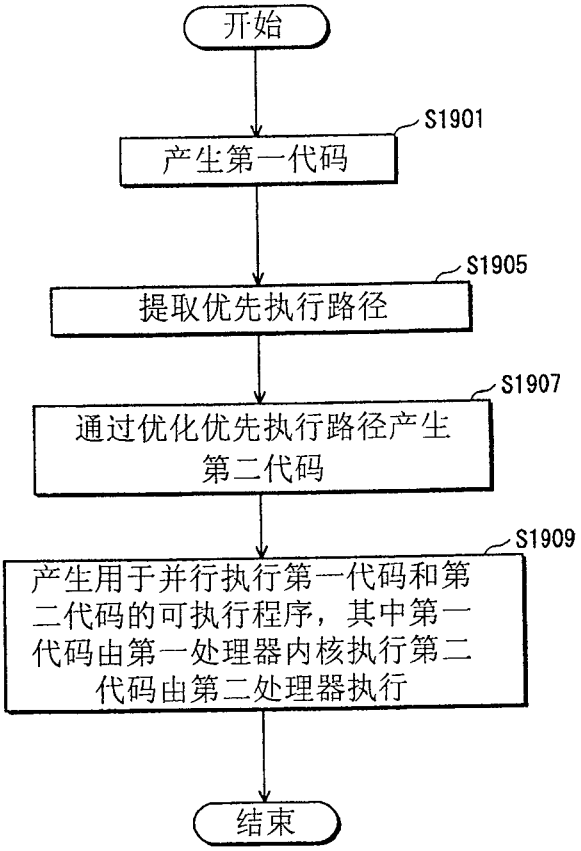

开始
S1901
产生第一代码
S1905
提取优先执行路径
S1907
通过优化优先执行路径产生第二代码
S1909
产生用于并行执行第一代码和第二代码的可执行程序，其中第一代码由第一处理器内核执行第二代码由第二处理器执行
结束

图 19

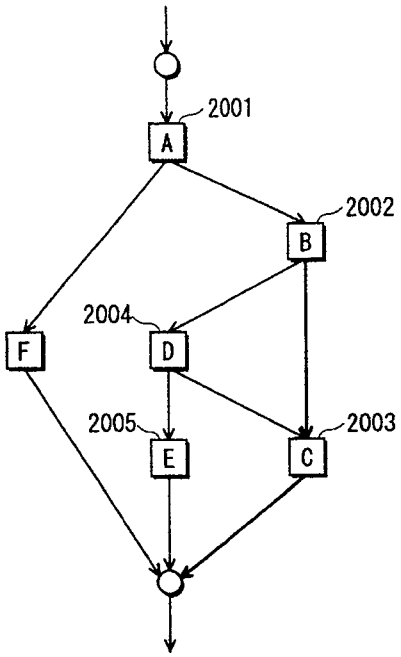

2001
A
2002
B
2004
F
D
2005
2003
E
C

图 20A

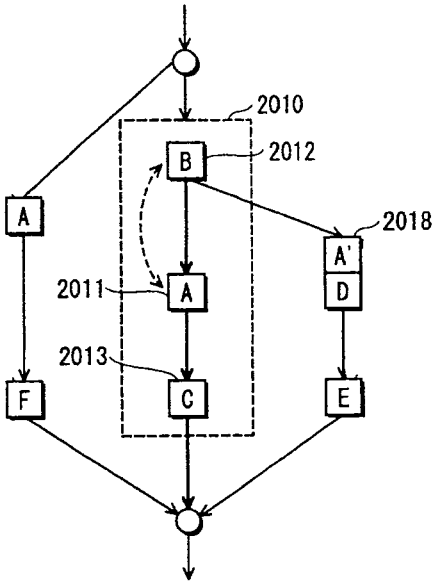

2010
B
2012
A
2018
A'
D
2011
A
2013
F
C
E

图 20B

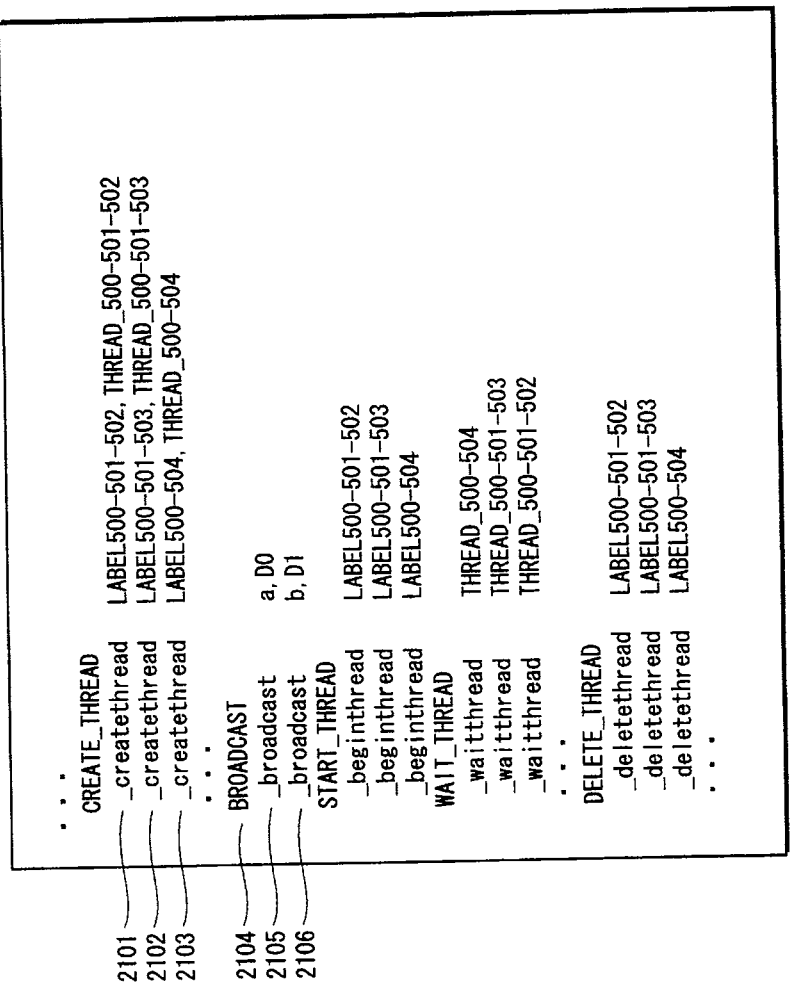

2100
. . .
CREATE_THREAD
_createthread LABEL500-501-502, THREAD_500-501-502
_createthread LABEL500-501-503, THREAD_500-501-503
_createthread LABEL500-504, THREAD_500-504
. . .
BROADCAST
_broadcast a, D0
_broadcast b, D1
START_THREAD
_beginthread LABEL500-501-502
_beginthread LABEL500-501-503
_beginthread LABEL500-504
WAIT_THREAD
_waitthread THREAD_500-504
_waitthread THREAD_500-501-503
_waitthread THREAD_500-501-502
. . .
DELETE_THREAD
_deletethread LABEL500-501-502
_deletethread LABEL500-501-503
_deletethread LABEL500-504
. . .
2101
2102
2103
2104
2105
2106

图 21